VHDL - Logique programmable

Partie 7
Les machines d'état

Denis Giacona

ENSISA
École Nationale Supérieure d'Ingénieurs Sud Alsace
12, rue des frères Lumière
68 093 MULHOUSE CEDEX
FRANCE

Tél. 33 (0)3 89 33 69 00

1. Le concept des machines d'états (FSM : Finite State Machine)

1.1. Propriétés des machines synchrones

- **Les sorties sont évaluées à partir d'un état interne qui englobe toute l'information relative au passé des entrées**
- **L'état interne est mémorisé dans le registre d'état (*flip-flops*)**
- **Un nouvel état interne se déduit de l'état interne présent et des entrées du système**
- **Dans une machine d'état synchrone toute évolution de l'état interne se produit sur le front d'une horloge**
- **La durée minimum d'un état interne est garantie (c'est un multiple de la période d'horloge)**

Tout système séquentiel synchrone peut être considéré comme une machine d'état ou une combinaison de machines d'état.

1.1.1. Machine de Moore (concerne uniquement les sorties)

- **Les sorties ne dépendent que de l'état courant**

E.F. Moore, "Gedanken-experiments on sequential machines", Automata Studies, Princeton University Press, 1956

1.1.2. Machine de Mealy (concerne uniquement les sorties)

- **Les sorties dépendent de l'état interne courant et des entrées**

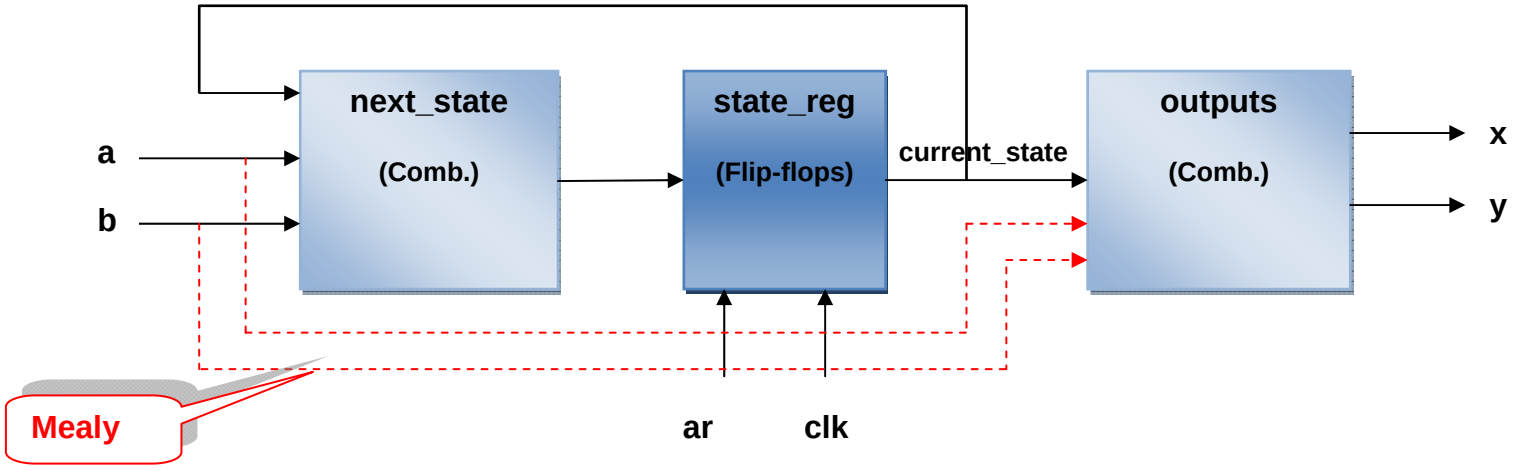
G.H. Mealy, "A method for synthesizing sequential circuits", Bell System Technical Journal, 1955

1.1.3. Machine mixte

- **Combinaison de sorties de type Mealy et de sorties de type Moore.**

1.2. Structure d'une machine de Moore ou de Mealy – sorties combinatoires

- **Principe : les sorties sont élaborées par un bloc combinatoire de décodage du registre d'état**
- **Inconvénients :**
  - **délai supplémentaire après le changement d'état interne de la machine**
  - **apparition en sortie d'impulsions parasites de courte durée dues aux hasards statiques (*glitches*) lorsque les couches combinatoires ont des longueurs différentes, et aux changements d'état transitoires des bits du registre d'état**



1.2.1. Sortie combinatoire – machine de Moore

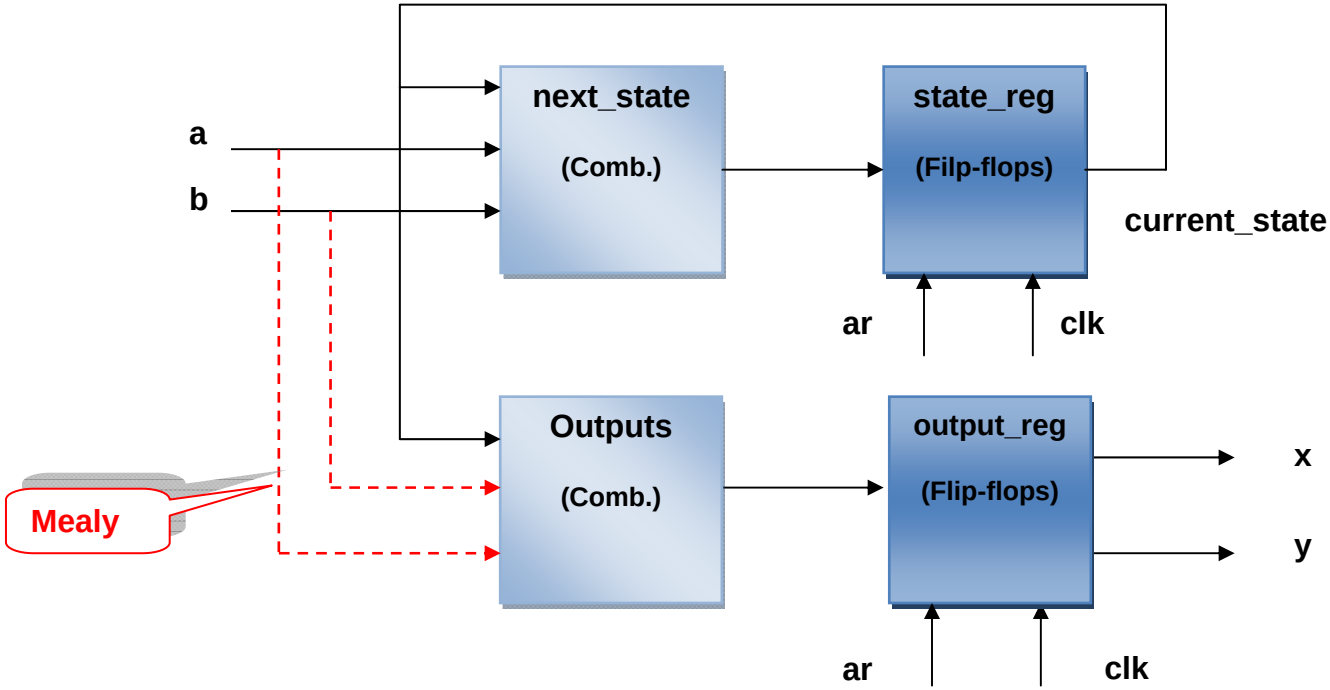
- **Les sorties ne changent pas d'état directement en réponse au changement d'état des entrées ; elles sont dépendantes du changement d'état de la machine (donc indirectement du front d'horloge)**

1.2.2. Sortie combinatoire – machine de Mealy

- **Les sorties peuvent changer directement en réponse au changement d'état des entrées, même entre 2 fronts d'horloge**

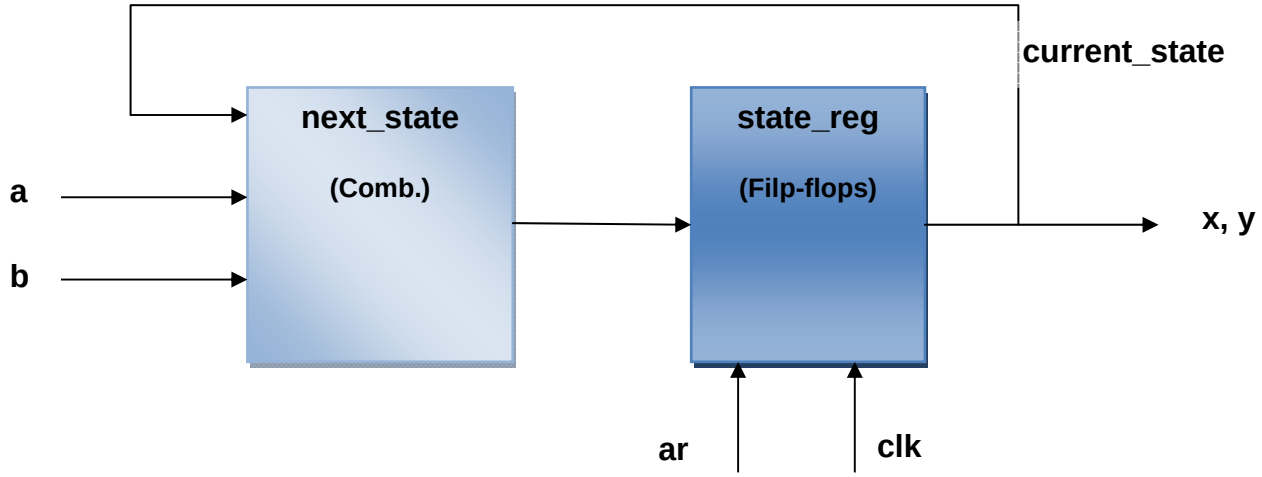
1.3. Structure d'une machine de Moore ou de Mealy – sorties synchronisées

- **Les sorties sont préparées avant le front actif d'horloge**
- **Les changements d'état des sorties ont tous lieu simultanément, sur le même front d'horloge**
- **Avantage : pas de hasard statique, indépendance vis-à-vis des changements d'état transitoires des bits du registre d'état**
- **Inconvénient : il faut un flip-flop supplémentaire par sortie**



1.4. Structure d'une machine de Moore – sorties prises directement dans l'état interne

- **Les bits du registre d'état fournissent directement les sorties**
- **Avantage : pas de combinatoire de sortie**
- **Inconvénient : méthode applicable uniquement dans des cas particuliers**



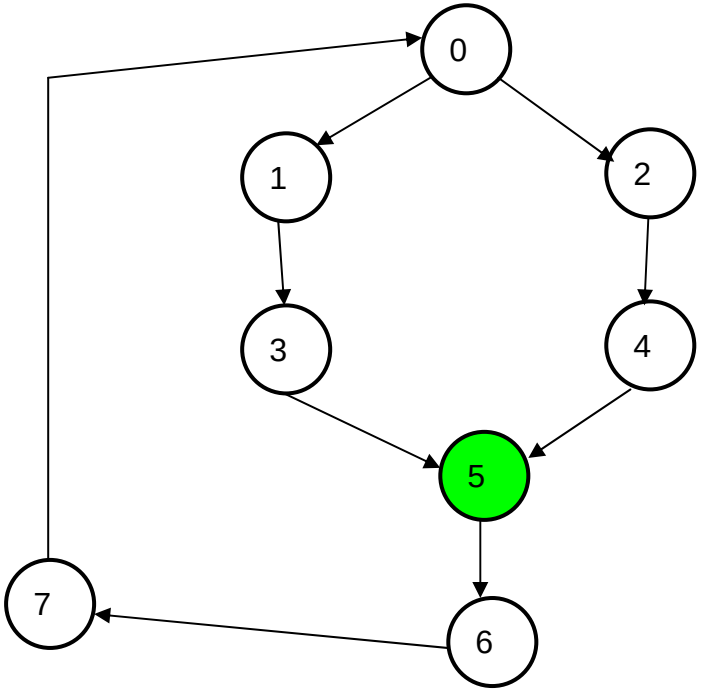
1.5. Codage des états internes de la machine

1.5.1. Choix du code

- **Binaire en séquence naturelle**
- **Gray**
- **Johnson**
- **One Hot** (un seul bit actif dans chaque combinaison)
- **Two Hot**
- **Autres (spécifiques)**

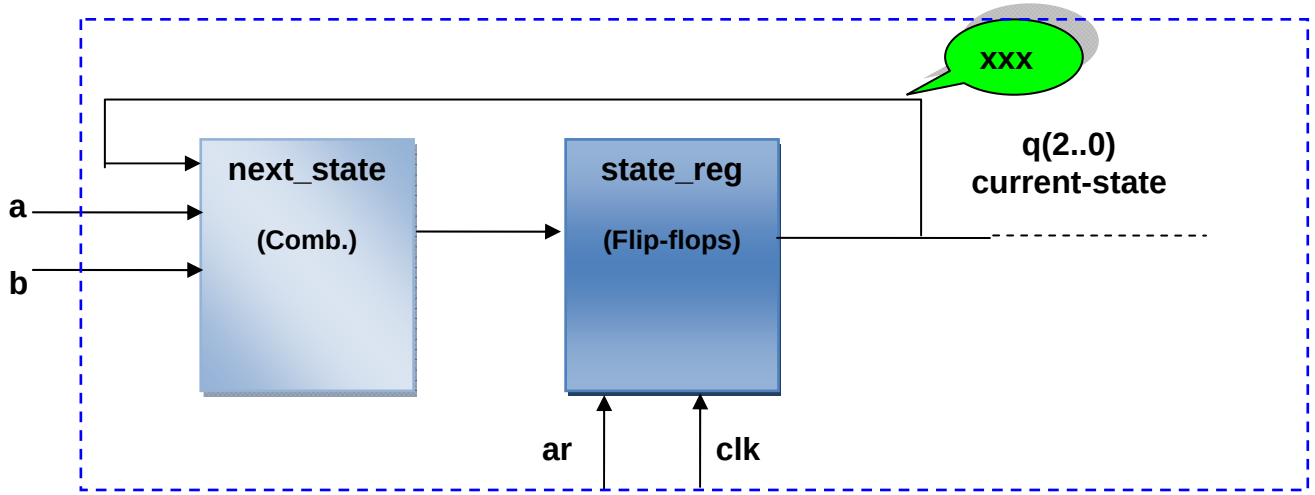
Par exemple pour coder 8 états :

Numéro état	*Binaire naturel*	*Gray*	*Johnson*	*One-hot*	*Autres*
0	000	000	0000	00000001	à définir
1	001	001	0001	00000010	à définir
2	010	011	0011	00000100	à définir
3	011	010	0111	00001000	à définir
4	100	110	1111	00010000	à définir
5	101	111	1110	00100000	à définir
6	110	101	1100	01000000	à définir
7	111	100	1000	10000000	à définir
	3 bits	*3 bits*	*4 bits*	*8 bits*	*à définir*



1.5.2. États codés

- **Chaque état interne est représenté par un code binaire dans lequel plusieurs bits sont à '1'**
- **La génération des sorties nécessite un post-décodage du registre d'état (logique combinatoire de sortie)**
- **Ils requièrent un minimum de flip-flops**
- **Si le nombre d'états est élevé, ils génèrent une logique combinatoire complexe pour le calcul de l'état futur**
- **C'est une méthode adaptée aux circuits CPLD (moins de flip-flops que dans les FPGA)**



☞ Code binaire naturel

- Le nombre de flip-flops « m » est minimal (jusqu'à 2^m états possibles)
- Il peut y avoir des états inutilisés ➔ prévoir des échappatoires (sinon risque de blocage)

☞ Code Johnson

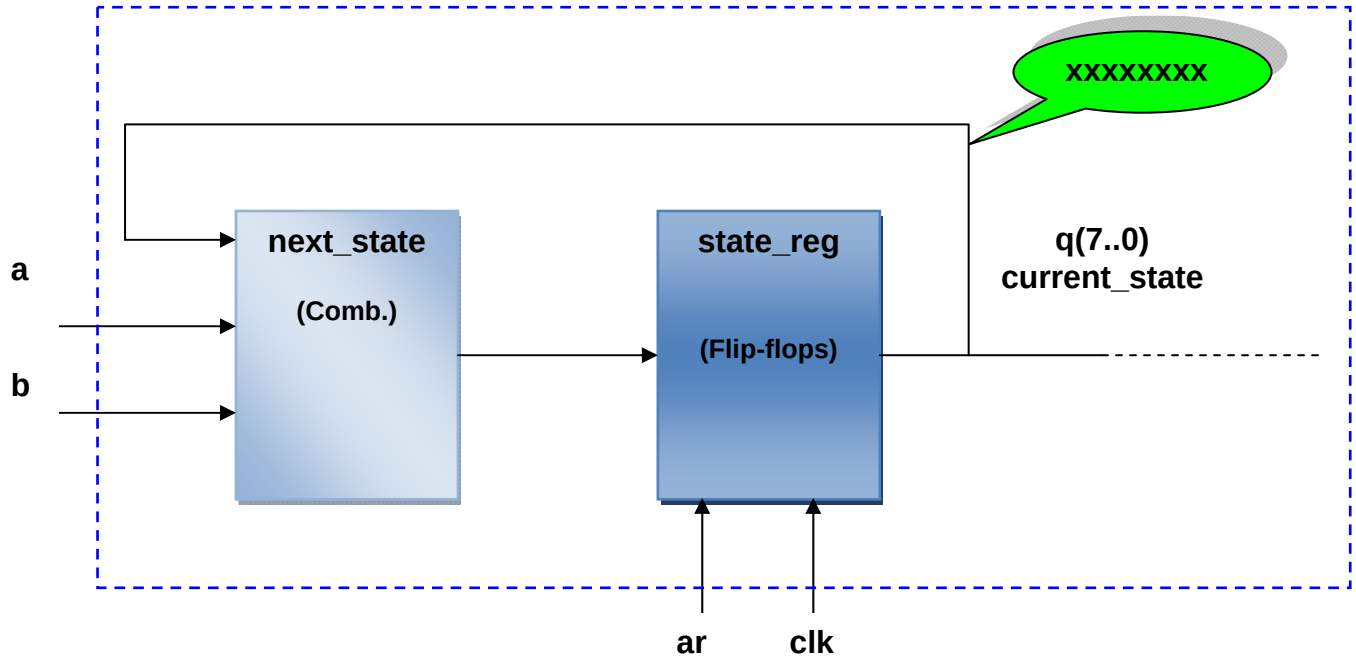
- Registre à décalage avec rétroaction complémentée
- Pour n flip-flops on peut coder 2n états
- Il comporte des états inutilisés

☞Code Gray (code binaire réfléchi)

- Le nombre de flip-flops « m » minimal (jusqu'à 2^m états)
- Un seul bit change de valeur entre deux états consécutifs ⇨ pas d'états transitoires
- Il peut y avoir des états inutilisés

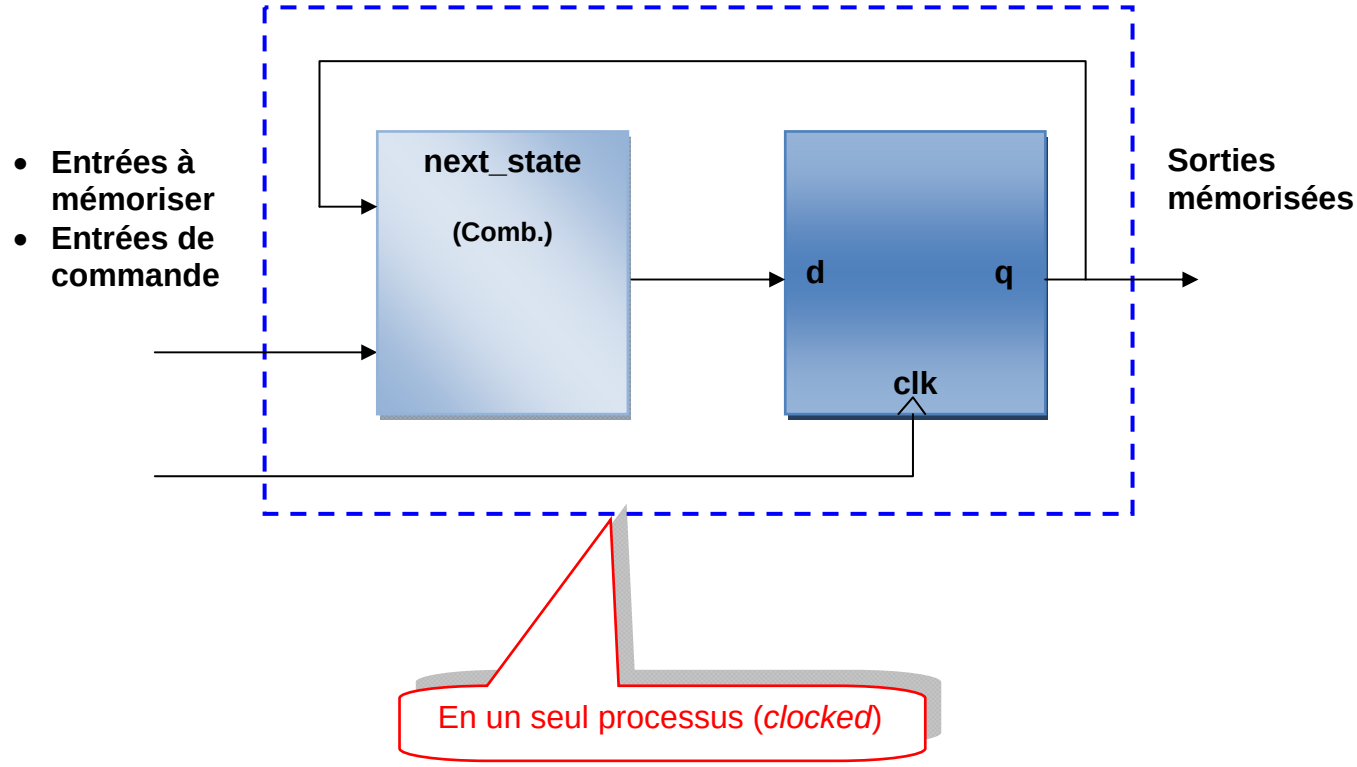
1.5.3. États décodés (One hot state machine)

- **Chaque état interne de la machine est représenté par un seul bit : c.-à-d. en fonctionnement normal, un seul flip-flop du registre d'état est à '1', les autres sont à '0'**
- **C'est une méthode adaptée aux circuits FPGA qui comportent un grand nombre de flip-flops**
- **La logique combinatoire de calcul de l'état futur est plus simple que pour les états codés**

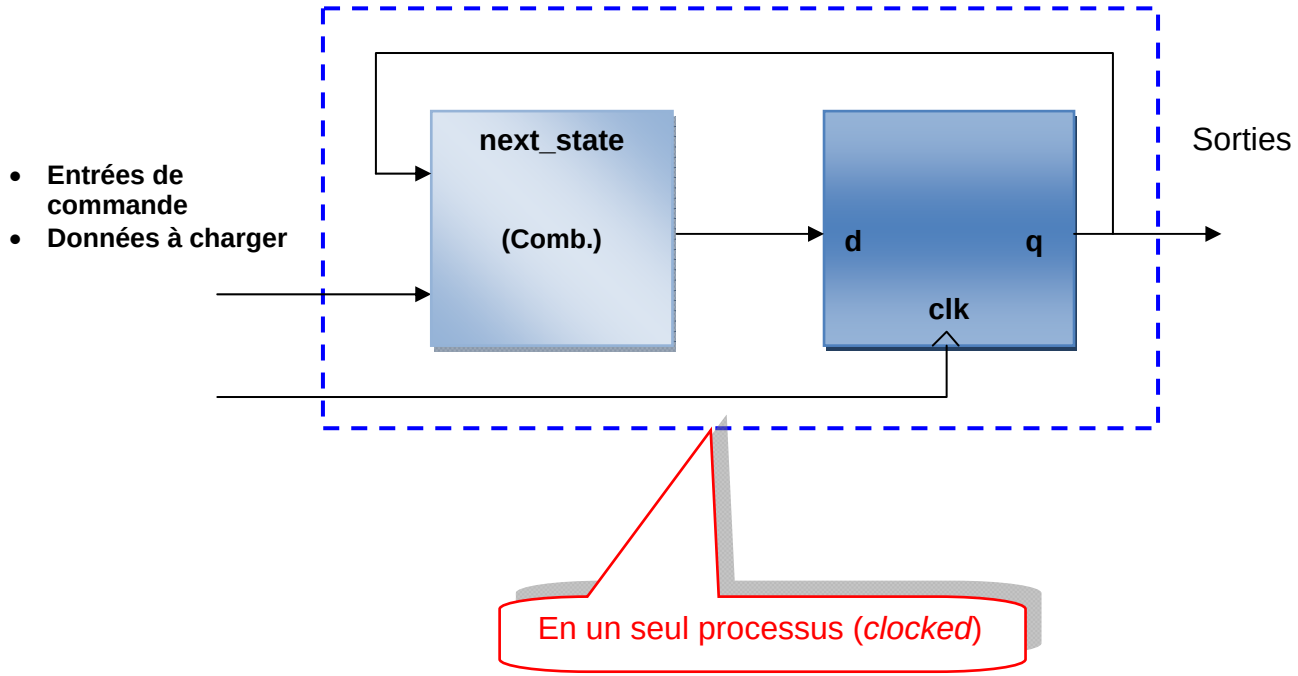


2. Description des machines d'états en VHDL

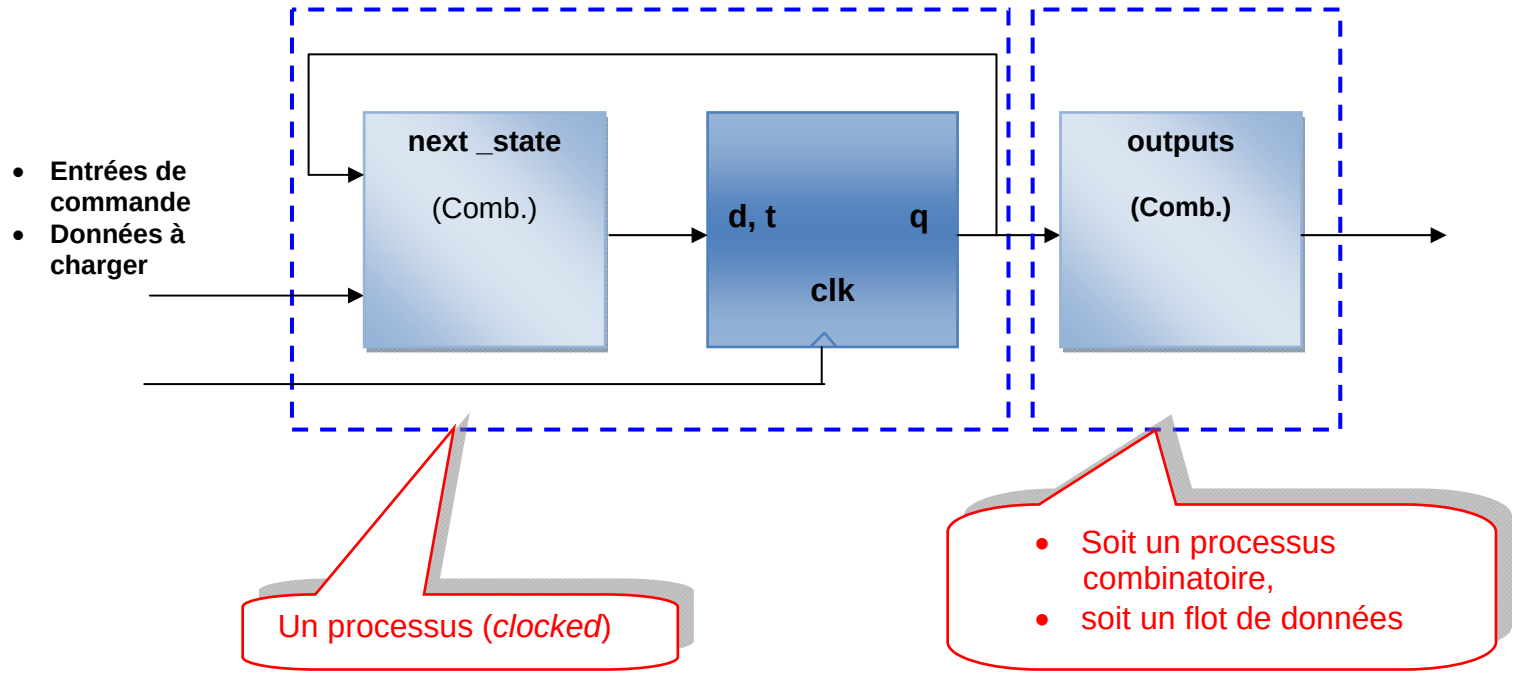
2.1. Les registres de mémorisation



2.2. Les registres à décalage



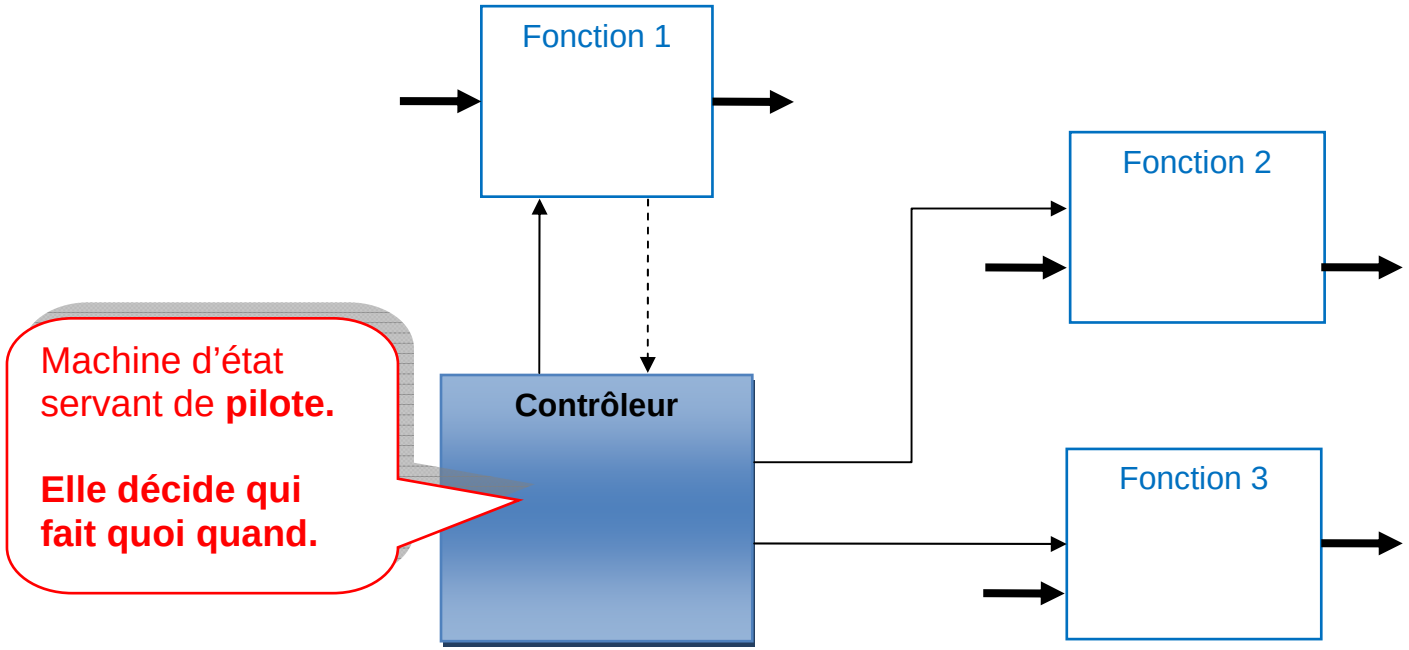
2.3. Les registres de comptage



Note : il est aussi possible de décrire le tout en un seul processus.

2.4. Les contrôleurs

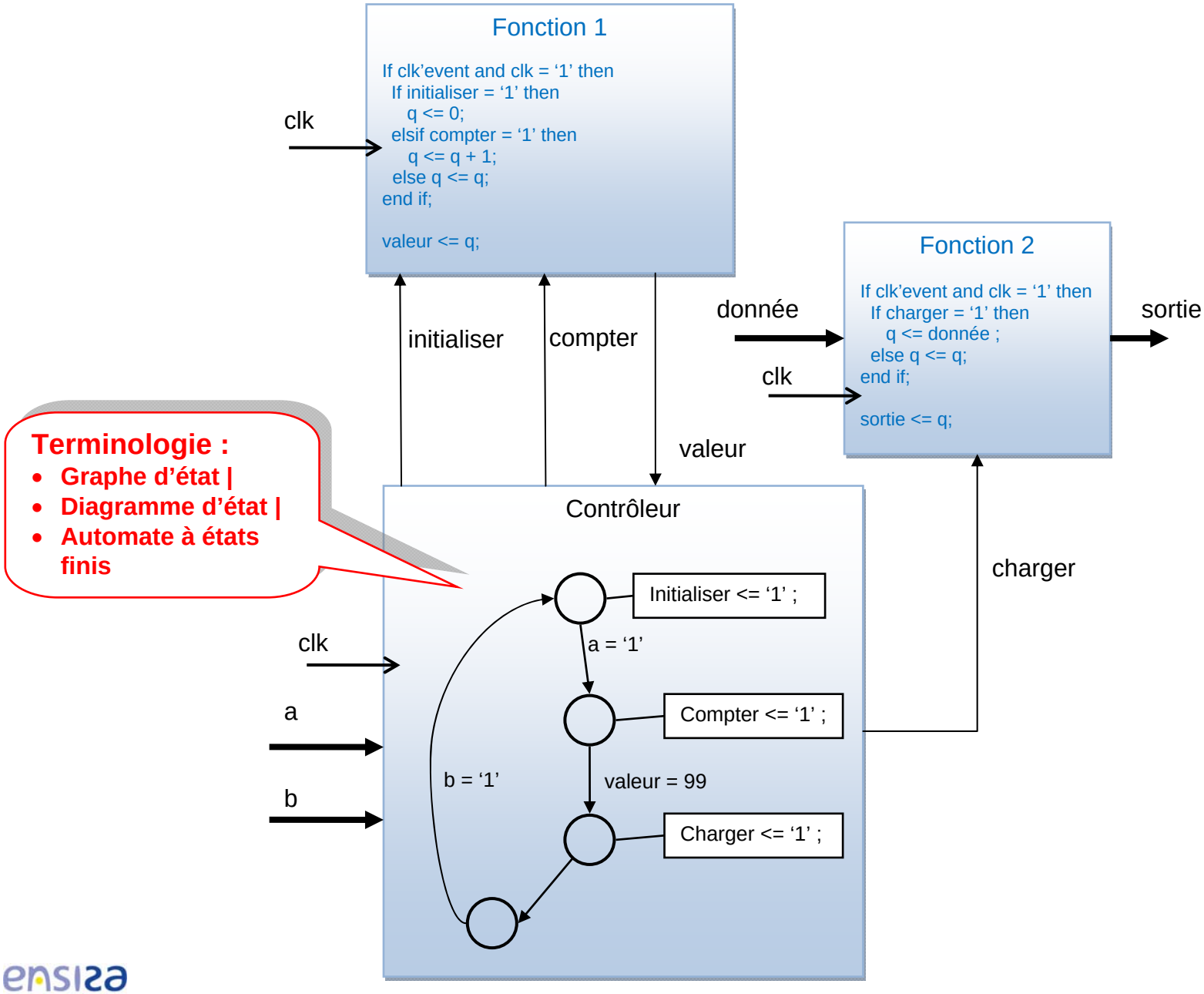
2.4.1. Rôle d'un contrôleur



Donnée

Ordre (activation, validation, déclenchement, arrêt, sélection)

Compte rendu


Fonction 1
If clk'event and clk = '1' then
If initialiser = '1' then
q <= 0;
elsif compter = '1' then
q <= q + 1;
else q <= q;
end if;
valeur <= q;
clk
initialiser
compter
valeur
Fonction 2
If clk'event and clk = '1' then
If charger = '1' then
q <= donnée ;
else q <= q;
end if;
sortie <= q;
donnée
clk
sortie
charger
Terminologie :
Graphe d'état |
Diagramme d'état |
Automate à états finis
Contrôleur
clk
a
b
Initialiser <= '1' ;
a = '1'
Compter <= '1' ;
valeur = 99
Charger <= '1' ;
b = '1'

2.4.2. Architecture VHDL d'un contrôleur

❑ Description avec 3 processus concurrents

```
-------------------------------------------------------------------
clocked_proc : PROCESS (clk, …)
-------------------------------------------------------------------

--      copie de next_state dans current_state sur front de clk
```

```
-------------------------------------------------------------------
nextstate_proc : PROCESS (current_state, …)
-------------------------------------------------------------------

--      calcul de next_state à partir de current_state et des entrées
```

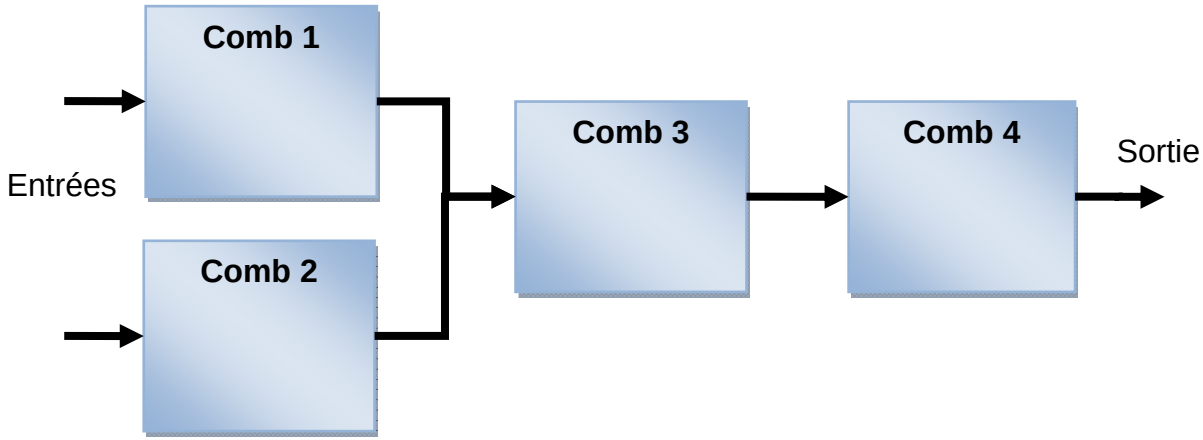
```
-------------------------------------------------------------------
output_proc : PROCESS (current_state, …)
-------------------------------------------------------------------

--      calcul des sorties à partir de current_state et éventuellement des entrées
```

3. Règles de conception pour un fonctionnement sûr

- **Une seule horloge si possible. Pas de décalages d'horloge (*clock skew*) sur les blocs séquentiels (dans les FPGA, utiliser les ressources de routage prévus pour la distribution de l'horloge)**
- **Synchroniser les entrées de données asynchrones**
  - **but : assurer la cohérence des signaux de commande du registre d'état**
- **Détecter ou corriger les erreurs consécutives à des événements extérieurs (radiations en milieu hostile : terrestres ou dans l'espace)**
  - **objectif : éviter les blocages de la machine d'état**
- **Ne pas appliquer les signaux combinatoires sujets aux hasards statiques sur les entrées asynchrones des blocs séquentiels (par exemple, ne pas fabriquer de signaux d'horloge avec des portes). Synchroniser si nécessaire les sorties**
- **Attendre la fin des transitoires avant d'exploiter les états internes des machines d'état**
- **S'assurer que les machines d'état démarrent dans des états valides. Synchroniser le signal servant à l'initialisation (*reset*)**
- **Choisir un codage adéquat pour le registre d'état**

3.1. Paramètres temporels et horloge

3.1.1. Systèmes combinatoires

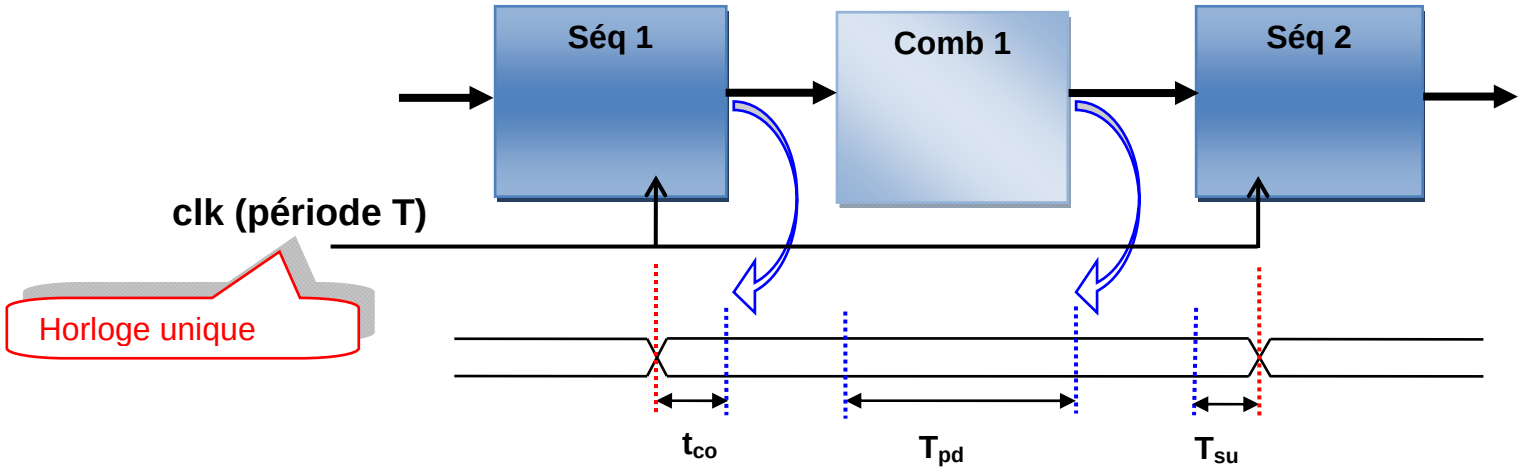


Disponibilité de la sortie :

à t = date d'arrivée des entrées + max(t_{pd1}, t_{pd2}) + t_{pd3} + t_{pd4}

t_{pd} : temps de propagation à travers un bloc combinatoire *(propagation delay)*

3.1.2. Système séquentiel (calcul de la fréquence max d'horloge)



Disponibilité de la sortie :

à T puis 2T puis 3T …

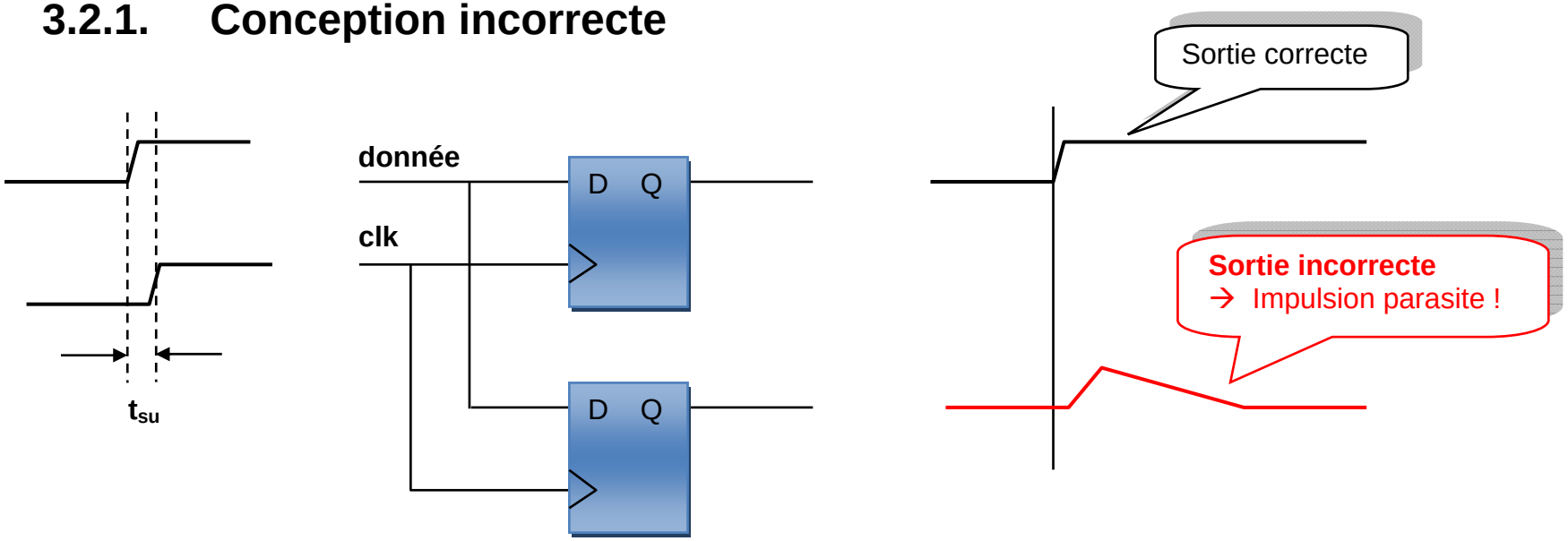
avec T = t_{co} + t_{pd1} + t_{su} + marge

t_{co} : *clock to output time*
t_{su} : *set up time*

3.2. La synchronisation des entrées asynchrones

☝ **Constat : Les états métastables existent, on ne peut pas les éviter !**

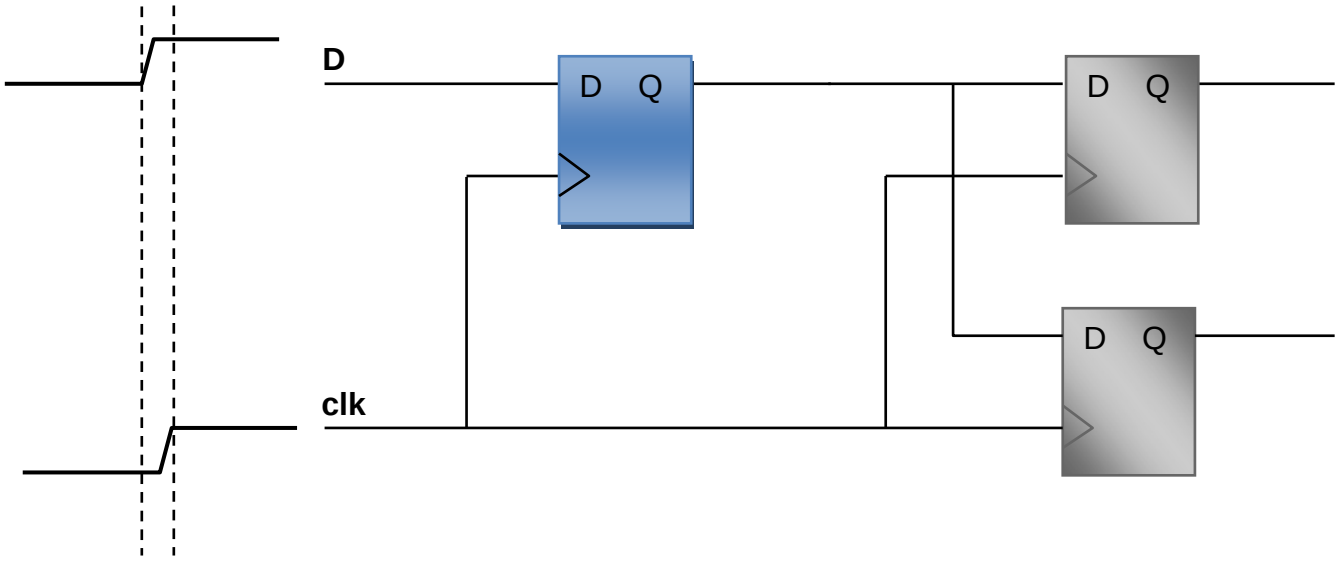
3.2.1. Conception incorrecte



Si l'un des deux flip-flops réagit bien et l'autre non (car ils peuvent avoir des caractéristiques physiques différentes, notamment s'ils appartiennent à des circuits intégrés différents), il y aura incohérence des signaux dans la couche logique suivante.

3.2.2. Conception correcte : synchronisation des entrées asynchrones

☺ REGLE : Tout niveau logique de rang 1 doit comporter un seul flip-flop D par entrée



- **Si le 1er flip-flop réagit mal, le signal est incorrect à sa sortie, mais il sera retombé à '0' au prochain front montant d'horloge, et les deux flips suivants n'auront pas vu l'impulsion transitoire parasite. Il y aura juste un retard supplémentaire d'une période d'horloge.**

- **Si le 1er flip-flop réagit bien, tant mieux !**

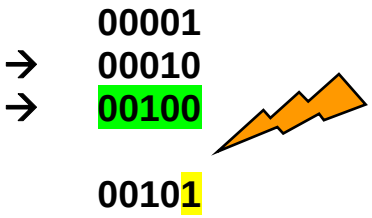
3.3. Les erreurs dues à des événements extérieurs

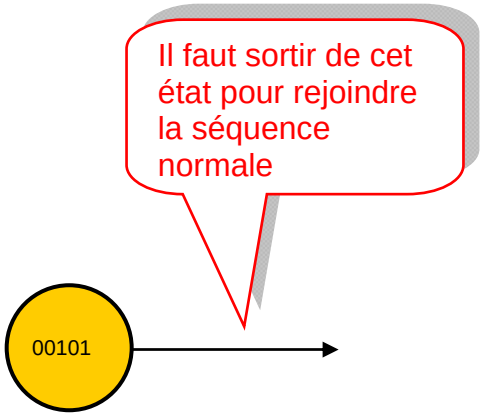
Des événements extérieurs peuvent faire commuter les flip-flops intempestivement et conduire la machine d'état dans un état ne figurant pas dans la séquence normale (*lockup states, illegal states*).

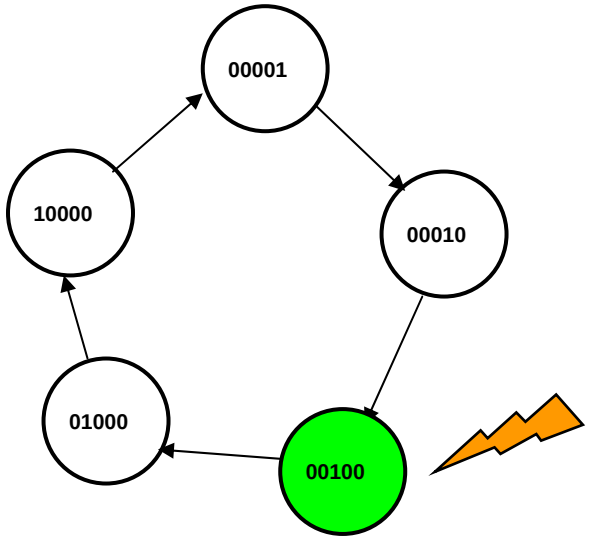
☺ REGLE
Pour un registre d'états à n bits, il faut faire une analyse en tenant compte des 2^n états possibles

3.3.1. Erreurs conduisant à des états inutilisés

- Machine à états décodés (*one hot FSM*)
- 5 états dans la séquence normale
- 5 bits, donc 32 – 5 = 27 combinaisons hors séquence normale



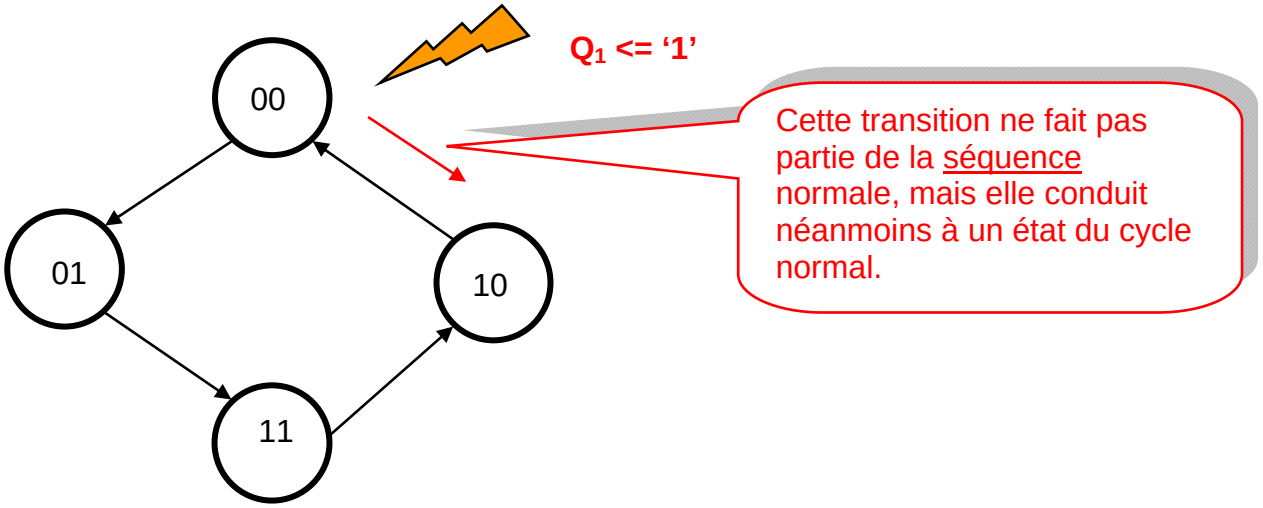




Cette erreur peut être détectée (deux flip-flops sont à '1'), mais ajouter de la logique ne résout pas forcément le problème, car cette logique supplémentaire est elle-même sensible aux perturbations extérieures !

3.3.2. Cas des machines comportant un code complet

- Machine à états codés (*Gray*)
- 4 états dans la séquence normale
- 2 bits, toutes les combinaisons sont utilisées

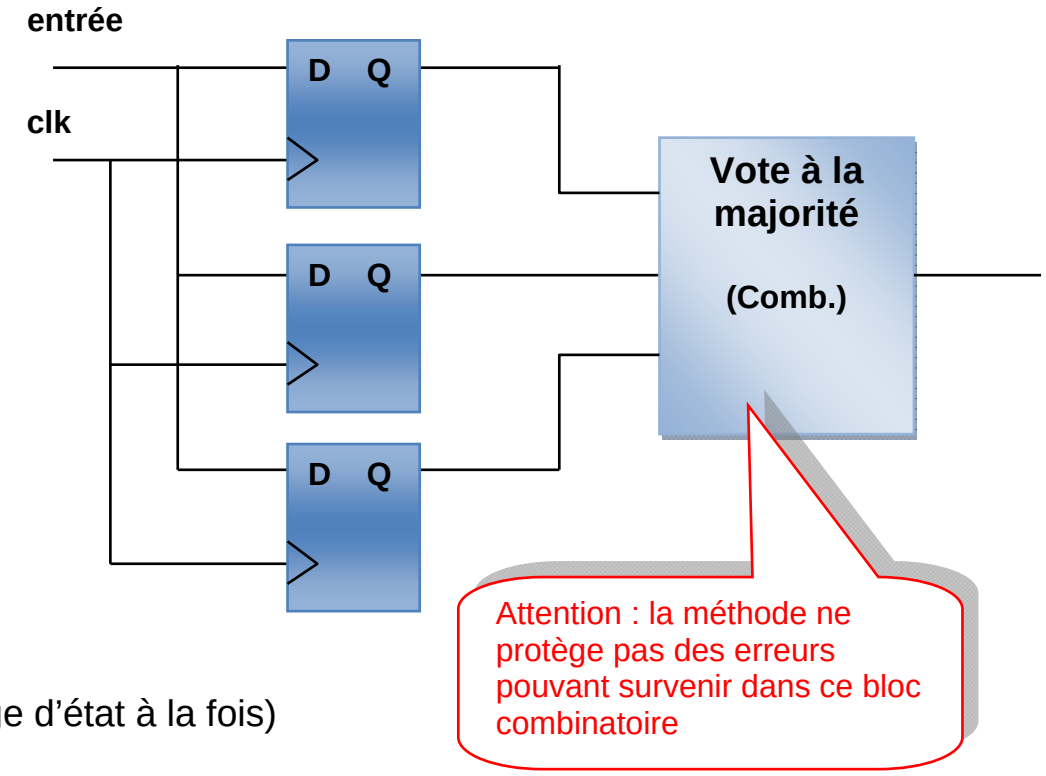


Il n'existe aucun moyen pour détecter cette erreur !

3.3.3. Méthodes de détection et de correction des erreurs

Redondance

- Deux systèmes identiques traitent la même donnée. Si les sorties sont différentes, on réinitialise le système
- Trois systèmes, ou plus, traitent la même donnée. On effectue un vote à la majorité.



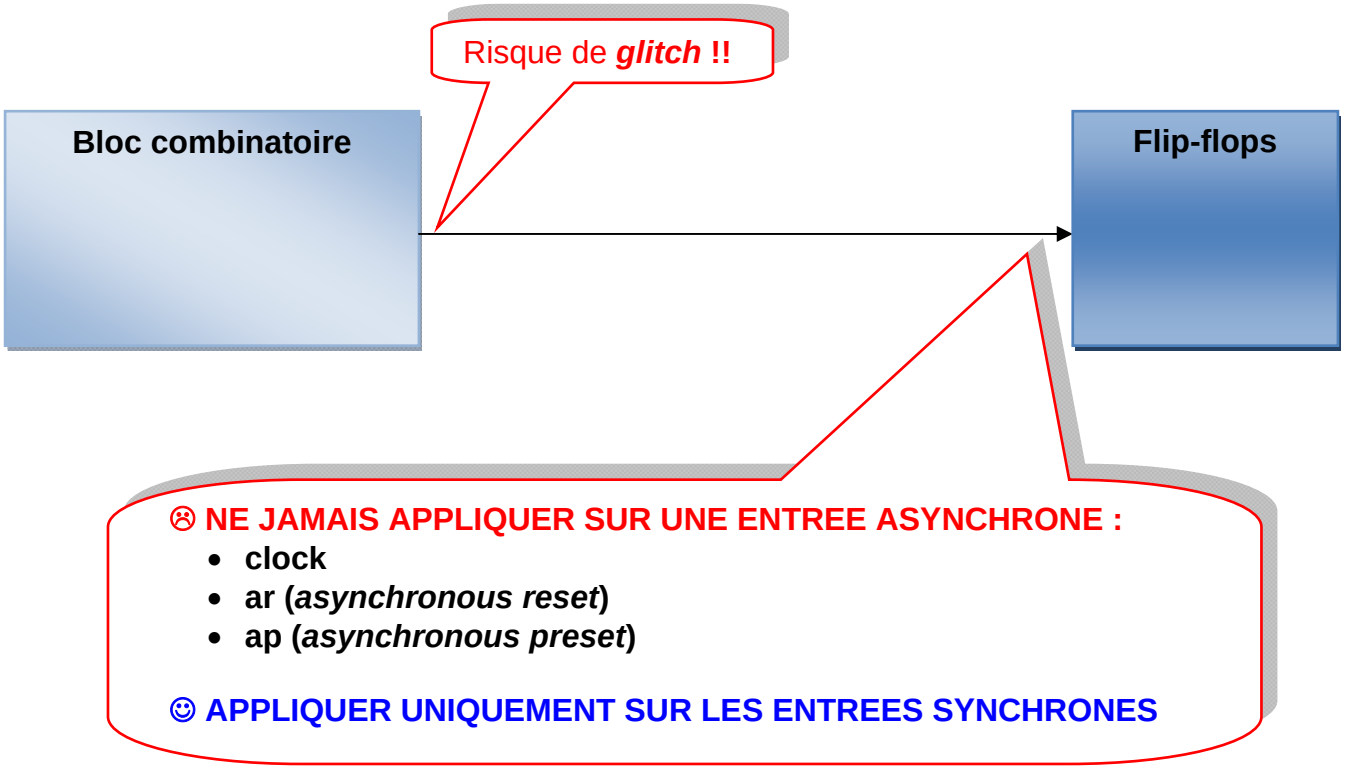
Exemple :

Chaque flip-flop de la conception initiale est remplacé par trois flip-flops (***TMR** : Triple Modular Redundancy*)

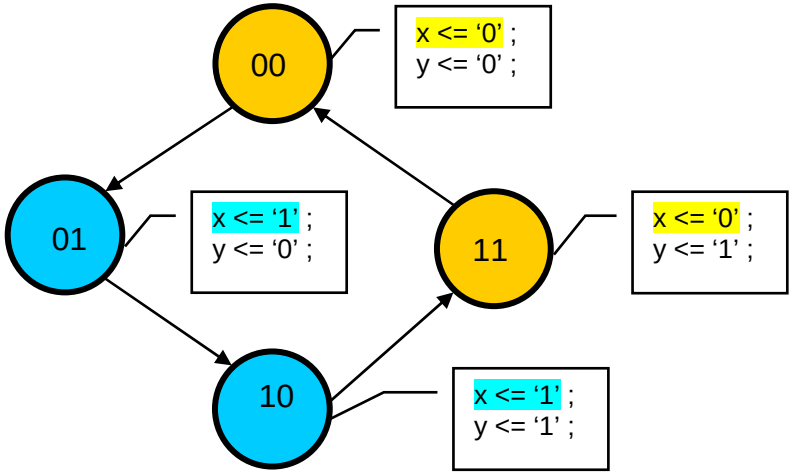
- **Parité**
  - Détection des erreurs simples (un seul bit change d'état à la fois)
- **Code de Hamming**
  - Détection des erreurs doubles et corrections des erreurs simples

3.4. Les hasards statiques

Les chemins de routage des circuits FPGA engendrent des délais de propagation pour les signaux. Ces délais peuvent provoquer des **hasards statiques** (*glitches*) à la sortie des blocs combinatoires, c.-à-d. des états logiques momentanément incorrects.



3.5. Les états internes transitoires



- En théorie, le passage de l'état **01** à l'état **10** change l'état de deux bits simultanément.
- En pratique, il peut y avoir un décalage entre les changements d'état des deux flip-flops.

État transitoire

Les transitions suivantes sont possibles : **01** → **11** → **10** ou bien **01** → **00** → **10**.

➔ Dans une machine de Moore à sorties combinatoires, **x** qui devrait rester à '1', pourrait prendre transitoirement une valeur fausse '0'.

4. Le modèle graphe d'état pour les machines synchrones

4.1. Les nœuds d'un graphe

4.1.1. Le cercle d'état

- Le cercle symbolise un état interne particulier de la machine
  - actif, il impose les actions sur les sorties
- Dans un graphe d'état connexe, un seul état est actif à la fois
- La description du comportement d'un système peut nécessiter plusieurs graphes d'état (synchronisés entre eux ou indépendants)

4.1.2. L'arc

- L'arc, aussi appelé transition, indique la possibilité de changement d'un état à un autre

4.2. La dynamique d'un graphe

4.2.1. L'horloge

- **L'horloge est implicite, elle ne figure jamais sur le graphe**
- **C'est toujours sur un front d'horloge (montant ou descendant) qu'un graphe peut franchir une transition, c.-à-d. passer d'un état à un autre**
- **Entre deux fronts d'horloge consécutifs, le graphe conserve son état interne**

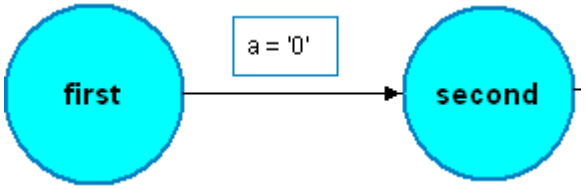
```
-------------------------------------------------
clocked_proc : PROCESS (clk,rst)
-------------------------------------------------
   BEGIN
      IF (rst = '1') THEN
         current_state <= first;
      ELSIF (clk'EVENT AND clk = '1') THEN
         current_state <= next_state;
      END IF;
   END PROCESS clocked_proc;
```

4.2.2. La transition avec condition

- La condition de transition est une expression booléenne comprenant des signaux d'entrée et/ou des signaux d'état internes ; elle participe à l'évolution du graphe

- L'état futur (*next state*) d'un graphe d'état dépend à la fois
  - de l'état courant (*current state*)
  - et de la condition de transition associée à un arc

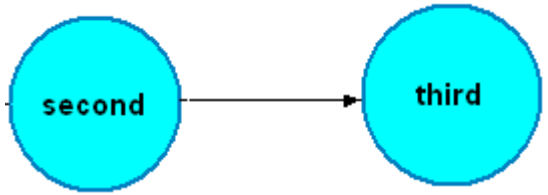
- Ci-contre, si `first` est l'état courant alors
  - si la condition `a = '0'` est vraie, au prochain front actif de l'horloge, le nouvel état sera second
  - sinon, il y a maintien (représentation implicite) dans l'état `first`




```
-------------------------------------------------------------------
   nextstate_proc : PROCESS (a, current_state)
-------------------------------------------------------------------
   BEGIN
     CASE current_state IS                    -- calcul de next_state en fonction de
        WHEN first =>                         --  current_state et de l'entrée a
            IF (a = '0') THEN
               next_state <= second;
            ELSE
               next_state <= first;
            END IF;
```

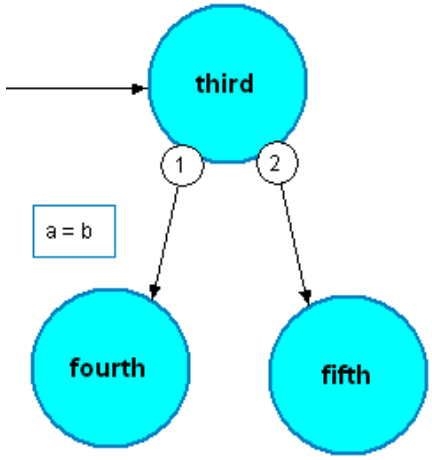
4.2.3. La transition directe

- **Il n'y a aucune inscription sur l'arc**
- **Ci-contre, l'état second ne dure qu'une période d'horloge, puis l'état `third` est atteint**




```
CASE current_state IS

WHEN second =>
            next_state <= third;
```

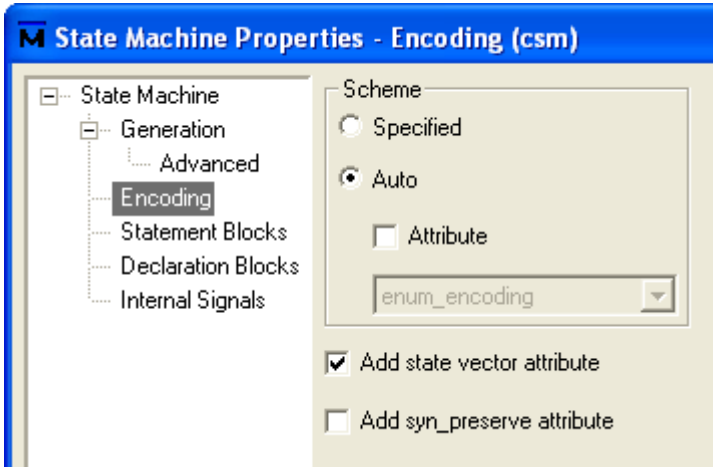
4.2.4. La transition multiple (structure OU à priorité)




```
CASE current_state IS


     WHEN third =>
         IF (a = b) THEN            -- priorité 1
            next_state <= fourth;
         ELSE                       -- priorité 2
            next_state <= fifth;
         END IF;
```

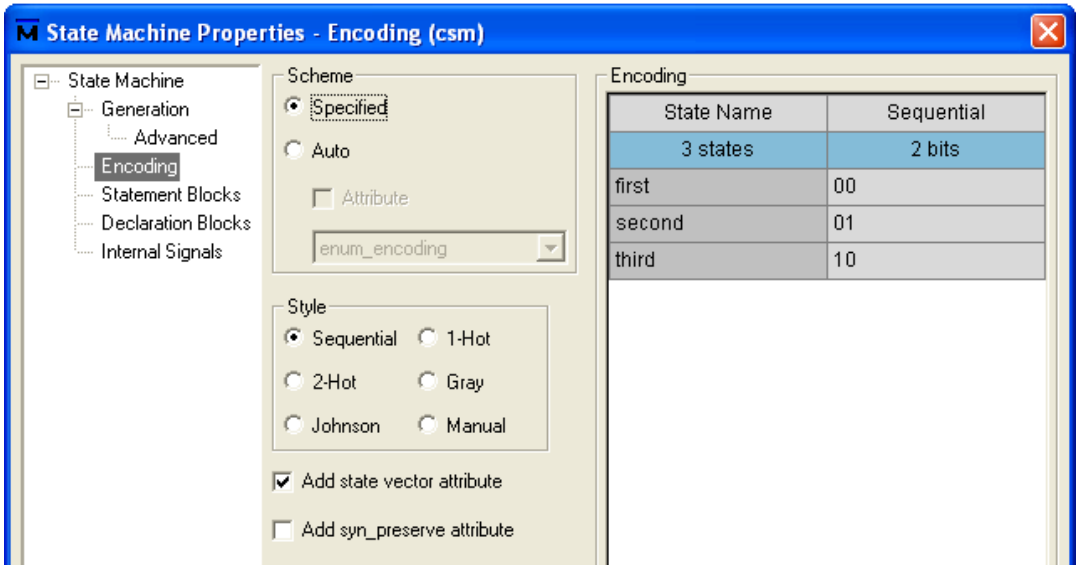
4.3. Le codage des états internes d'un graphe

4.3.1. États de type énuméré



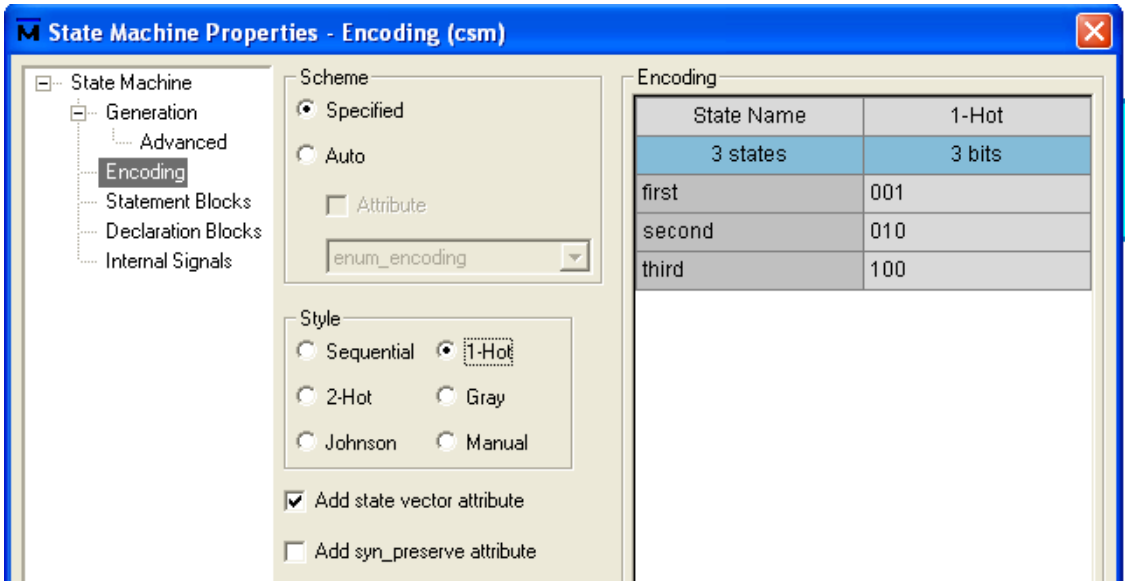

```
TYPE STATE_TYPE IS (first, second, third);

-- State vector declaration
ATTRIBUTE state_vector : string;
ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state";

-- Declare current and next state signals
SIGNAL current_state : STATE_TYPE;
SIGNAL next_state : STATE_TYPE;
```

4.3.2. États codés en binaire



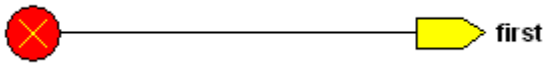

```
SUBTYPE STATE_TYPE IS  std_logic_vector(1 DOWNTO 0);
   -- State vector declaration
   ATTRIBUTE state_vector : string;
   ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state";

   -- Hard encoding
   CONSTANT first : STATE_TYPE := "00";
   CONSTANT second : STATE_TYPE := "01";
   CONSTANT third : STATE_TYPE := "10";

   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
```

4.3.3. États codés « one hot »




```
SUBTYPE STATE_TYPE std_logic_vector(2 DOWNTO 0);
    -- State vector declaration
   ATTRIBUTE state_vector : string;
   ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state";
   -- Hard encoding
   CONSTANT first : STATE_TYPE := "001";
   CONSTANT second : STATE_TYPE := "010";
   CONSTANT third : STATE_TYPE := "100";
   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
```

4.4. L'état de repli d'un graphe

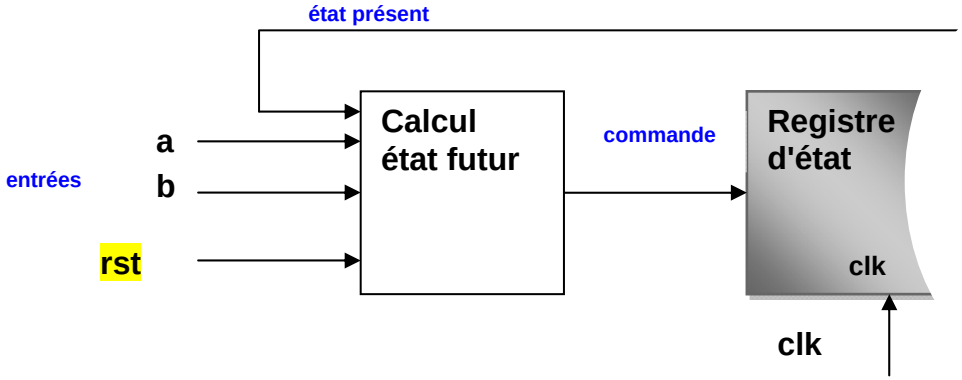



```
nextstate_proc : PROCESS (a,current_state)

   BEGIN
      CASE current_state IS
         WHEN first =>
            IF (a = '0') THEN
               next_state <= second;
            ELSE
               next_state <= first;
            END IF;
         WHEN second =>
            next_state <= third;
         WHEN third =>
            next_state <= third;
         WHEN OTHERS =>
            next_state <= first;
      END CASE;
   END PROCESS nextstate_proc;
```

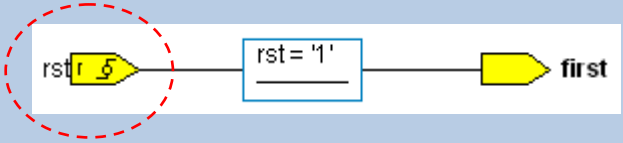
4.5. L'initialisation d'un graphe

4.5.1. Reset synchrone

- Le signal `rst` est traité comme une entrée normale
- Si la condition d'initialisation `rst` est vraie, alors au prochain front actif d'horloge le système entre dans l'état initial `first`

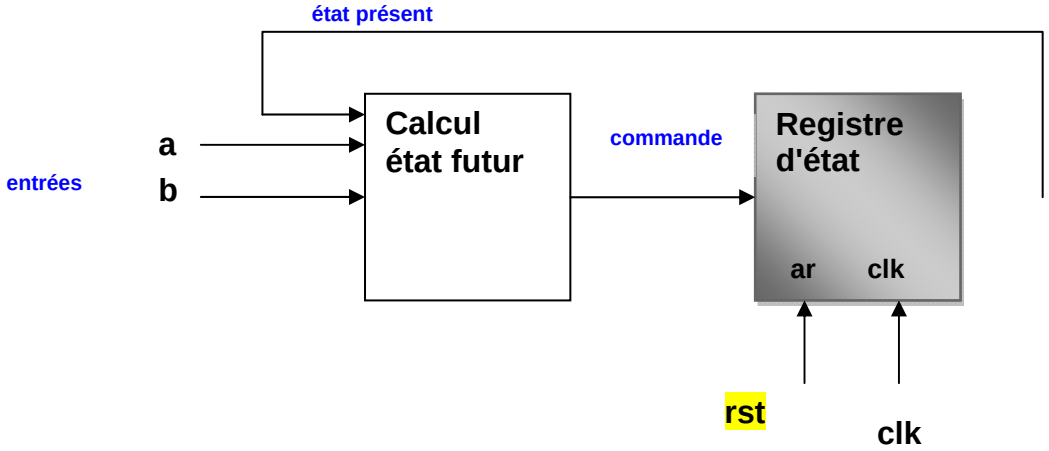



```
BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         IF (rst = '1') THEN
            current_state <= first;
         ELSE
            current_state <= next_state;
         END IF;
      END IF;
END PROCESS clocked_proc;
```

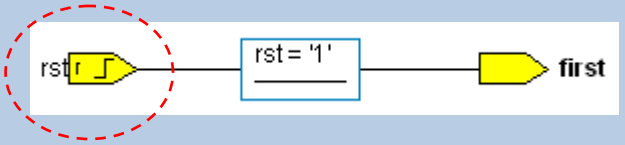


4.5.2. Reset asynchrone

- **Le signal `rst` agit directement sur les entrées `ar` (*asynchronous reset*) des flip-flops**




```
BEGIN
      IF (rst = '1') THEN
         current_state <= first;
      ELSIF (clk'EVENT AND clk = '1') THEN
         current_state <= next_state;
      END IF;
END PROCESS clocked_proc;
```



4.6. Les états particuliers de l'outil ModelSim Designer

4.6.1. L'état d'attente

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

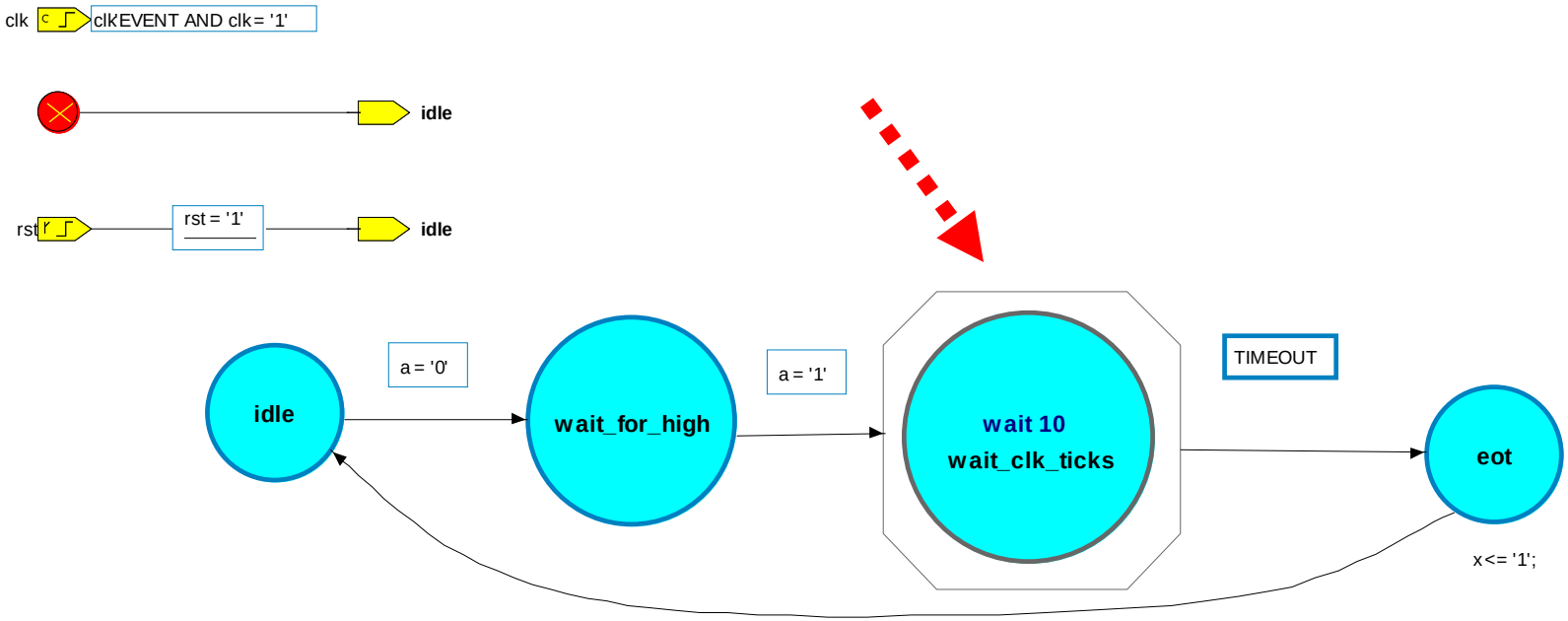
SIGNAL	MODE	DEFAULT	RESET	SCHEME
x	OUT	'0'		COMB

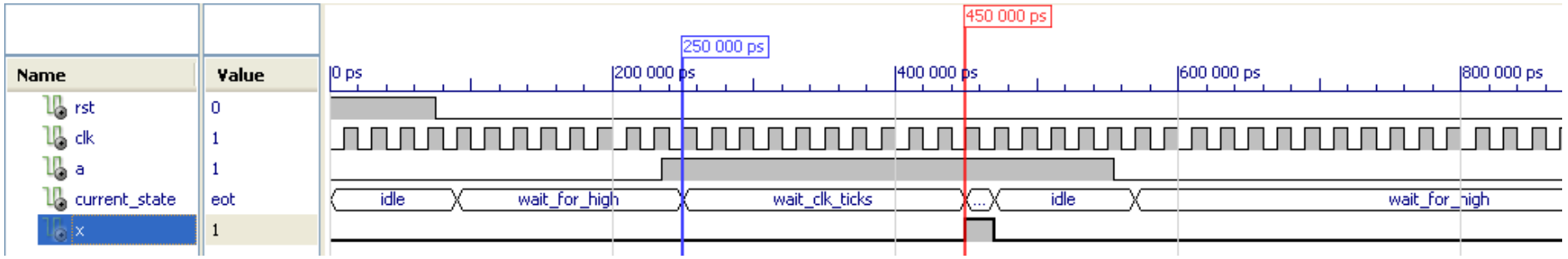
State Register Statements

Process Declarations
Clocked Process:
Output Process:

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

<company name>		Project:	hierarchy_state
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/fsm/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 15 févr. 2010		




Name
Value
rst
0
clk
1
a
1
current_state
eot
x
1
0 ps
200 000 ps
250 000 ps
400 000 ps
450 000 ps
600 000 ps
800 000 ps
idle
wait_for_high
wait_clk_ticks
idle
wait_for_high

4.6.2. L'état hiérarchique

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

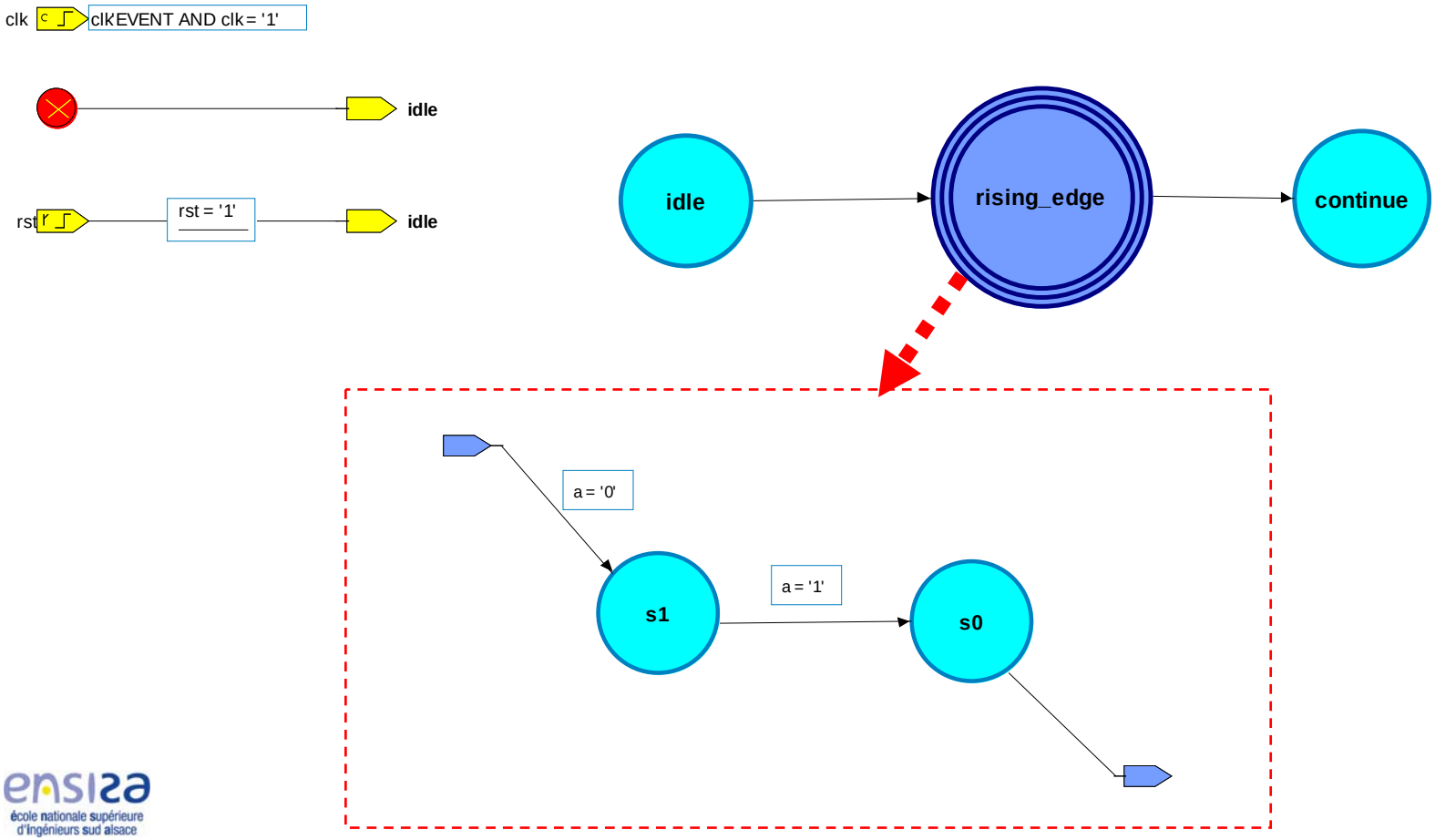
SIGNAL	MODE	DEFAULT	RESET	SCHEME
x	OUT	'0'		COMB

State Register Statements

Process Declarations
Clocked Process:
Output Process:

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

<company name>		Project:	hierarchy_state
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/fsm/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 15 févr. 2010		



4.7. Les actions

- **Une action est un ordre permettant**
  - **soit d'assigner des valeurs '0' ou '1' à des ports de sortie**
  - **soit d'initier des opérations complexes**
    - **Comptage**
    - **Décomptage**
    - **Décalage à droite**
    - **Décalage à gauche**
    - **Mémorisation**
    - **Opérations arithmétiques**
    - **Multiplexage**

- **Modes d'action**
  - **action d'état** **(machine de Moore)**
  - **action de transition** **(machine de Mealy)**

- **Modes d'assignation d'une sortie**
  - **assignation par un bloc combinatoire**
  - **assignation par un registre**

4.7.1. Action d'état – assignation combinatoire de la sortie

Global Actions **Concurrent Statements** **Architecture Declarations** **Signal Status** **State Register Statements** **Process Declarations**

Pre Actions:

Post Actions:

SIGNAL	MODE
x	OUT

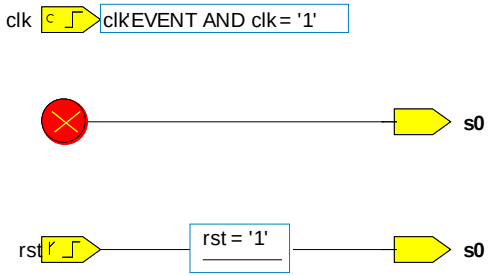
DEFAULT	RESET	SCHEME
'0'		COMB

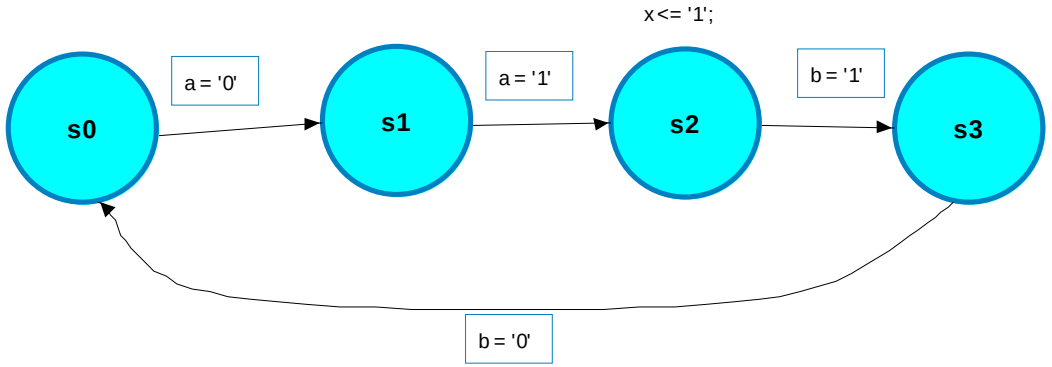
Clocked Process:

Output Process:

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```





<company name>		Project:	moore_comb
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/fsm/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 05 févr. 2010		

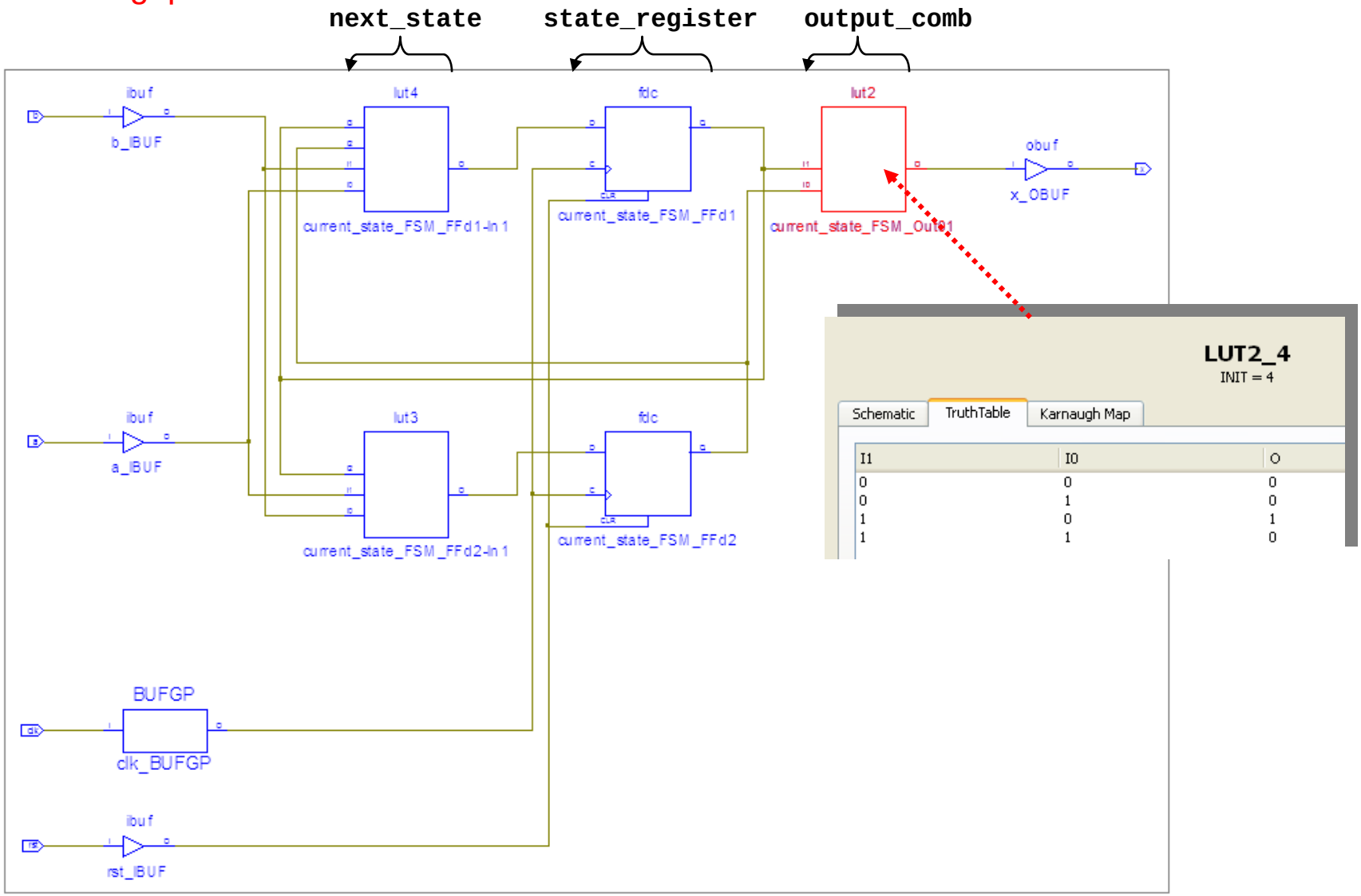
	A	B	C	D	E	F	G	H	I	J	K	L	M
A	Group	Name	Mode	Type	Bounds	Initial	Category	Assign In	Expression	Scheme	Default	Reset	Comment
1		a	IN	std_logic			Data						
2		b	IN	std_logic			Data						
3		clk	IN	std_logic			Clock (Rising)		clk'EVENT AND clk = '1'				
4		rst	IN	std_logic			Reset (Async High)		rst = '1'				
5		x	OUT	std_logic			Data	<auto>		Comb	'0'		

```
output_proc : PROCESS (current_state)

   BEGIN
      -- Default Assignment
      x <= '0';

      -- Combined Actions
      CASE current_state IS
         WHEN s2 =>
            x <= '1';
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS output_proc;
```

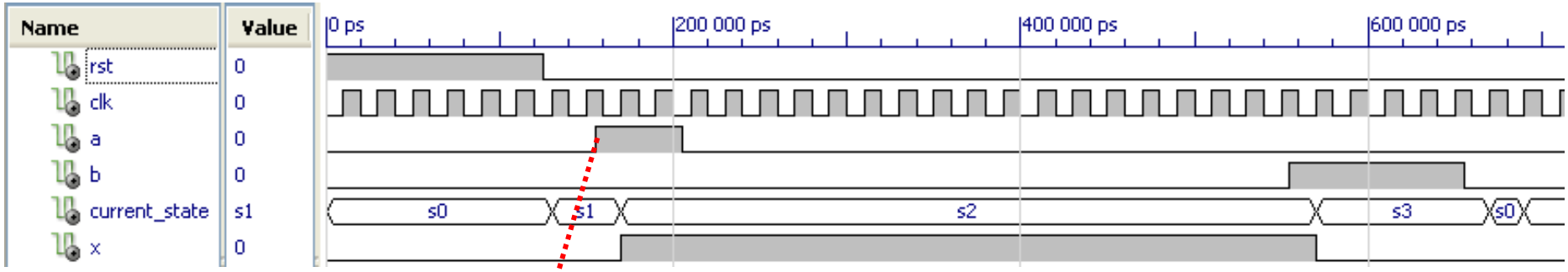
La sortie **x**

- ne dépend que de l'état interne de la machine d'état
- passe à '1' dès que la machine entre dans l'état **s2** (au retard près dû à la traversée du bloc combinatoire de sortie)
- reste à '1' durant tout le séjour dans l'état **s2**
- est à '0' par défaut lorsqu'elle n'est pas assignée explicitement dans un état du graphe

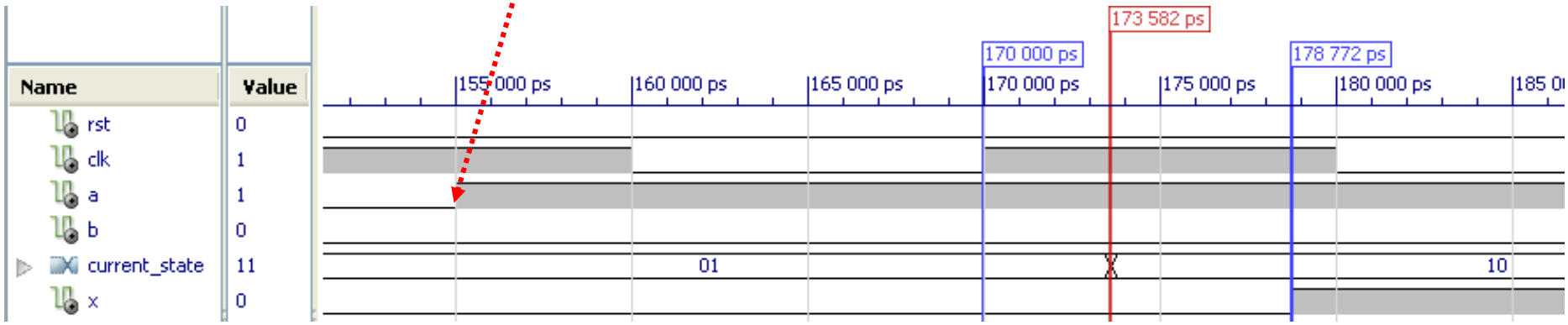
❑ Schéma technologique



❑ Simulation comportementale



❑ Simulation temporelle



Temps de commutation des flip-flops

Délai de propagation du bloc combinatoire de sortie

4.7.2. Action d'état – assignation registre de la sortie

Global Actions **Concurrent Statements** **Architecture Declarations** **Signal Status** **State Register Statements** **Process Declarations**

Pre Actions:

Post Actions:

SIGNAL	MODE	DEFAULT	RESET	SCHEME
x	OUT	'0'	'0'	REG

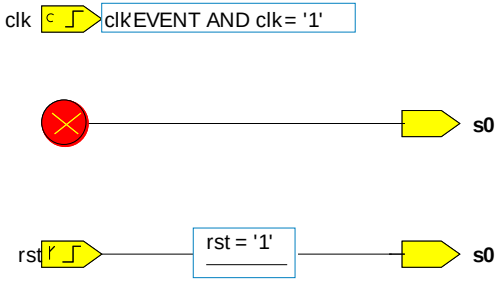
Clocked Process:

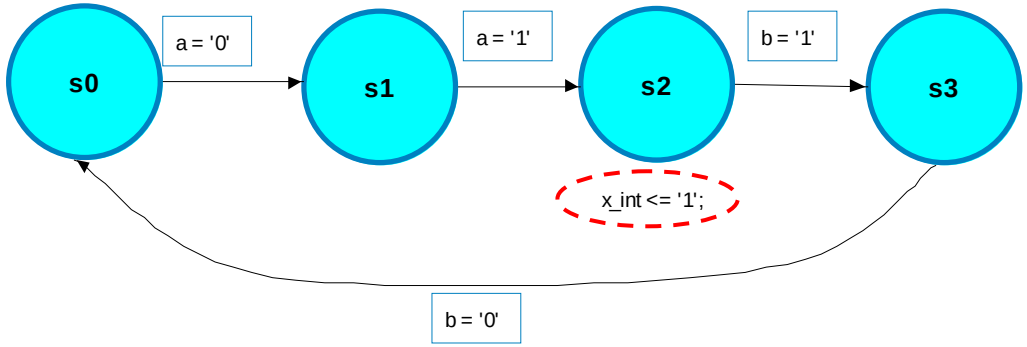
Output Process:

Package List

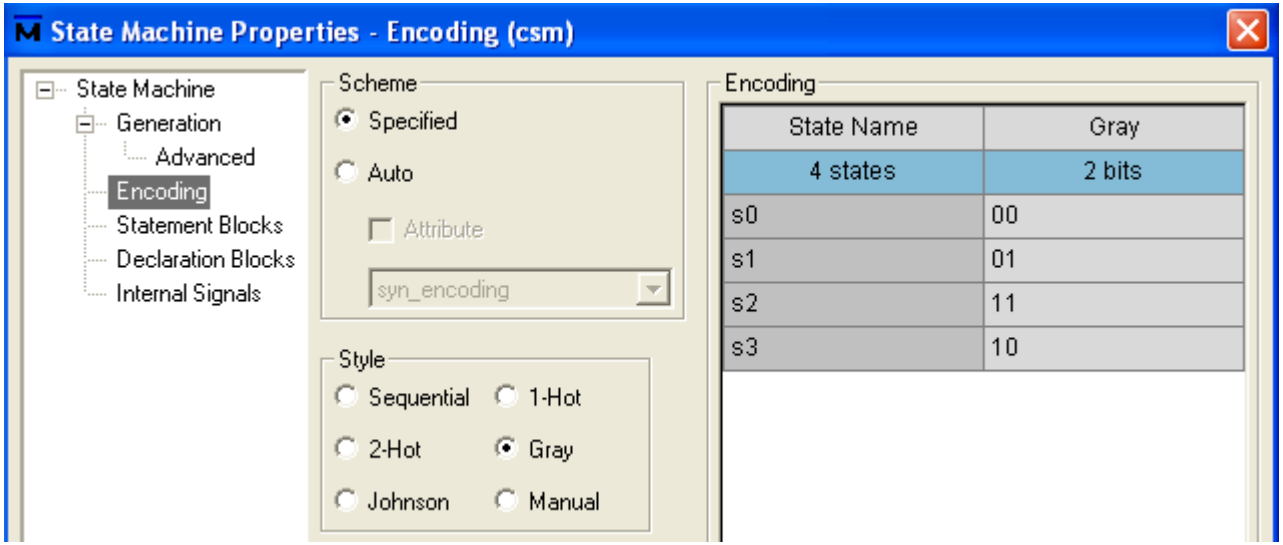
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

L'éditeur remplace le signal de sortie **x** par un signal interne **x_int**





<company name>		Project:	moore_reg
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	work/fsm/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 05 févr. 2010		




```
ARCHITECTURE fsm OF fsm IS

   SUBTYPE STATE_TYPE IS  std_logic_vector(1 DOWNTO 0);

   -- Hard encoding
   CONSTANT s0 : STATE_TYPE := "00";
   CONSTANT s1 : STATE_TYPE := "01";
   CONSTANT s2 : STATE_TYPE := "11";
   CONSTANT s3 : STATE_TYPE := "10";

   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
```

```
output_proc : PROCESS (current_state)
   BEGIN
      -- Default Assignment
      x_int <= '0';
      -- Combined Actions
      CASE current_state IS
         WHEN s2 =>
            x_int <= '1';
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS output_proc;
```

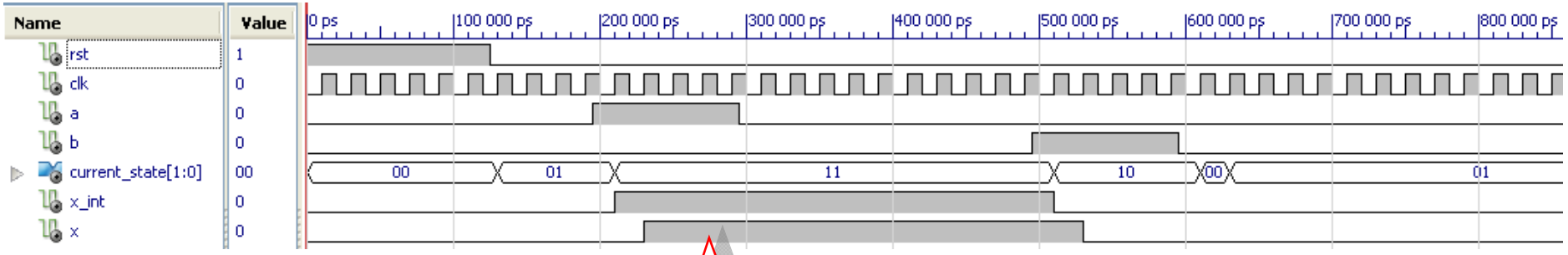
Le signal interne **x_int**
- ne dépend que de l'état interne de la machine d'état
- passe à '1' dès que la machine entre dans l'état **s2**
- reste à '1' durant tout le séjour dans l'état **s2**
- est à '0' par défaut lorsqu'il n'est pas assigné explicitement dans un état du graphe

```
clocked_proc : PROCESS (clk, rst)
BEGIN
      IF (rst = '1') THEN
         current_state <= s0;
         -- Default Reset Values
         x <= '0';
      ELSIF (clk'EVENT AND clk = '1') THEN
         current_state <= next_state;
         -- Registered output assignments
         x <= x_int;
      END IF;
   END PROCESS clocked_proc;
```

La sortie **x**
- prend la valeur de **x_int** sur un front d'horloge

  ⇨ la sortie **x** est en retard d'une période d'horloge par rapport à l'entrée dans l'état interne **s2**

❑ Simulation comportementale

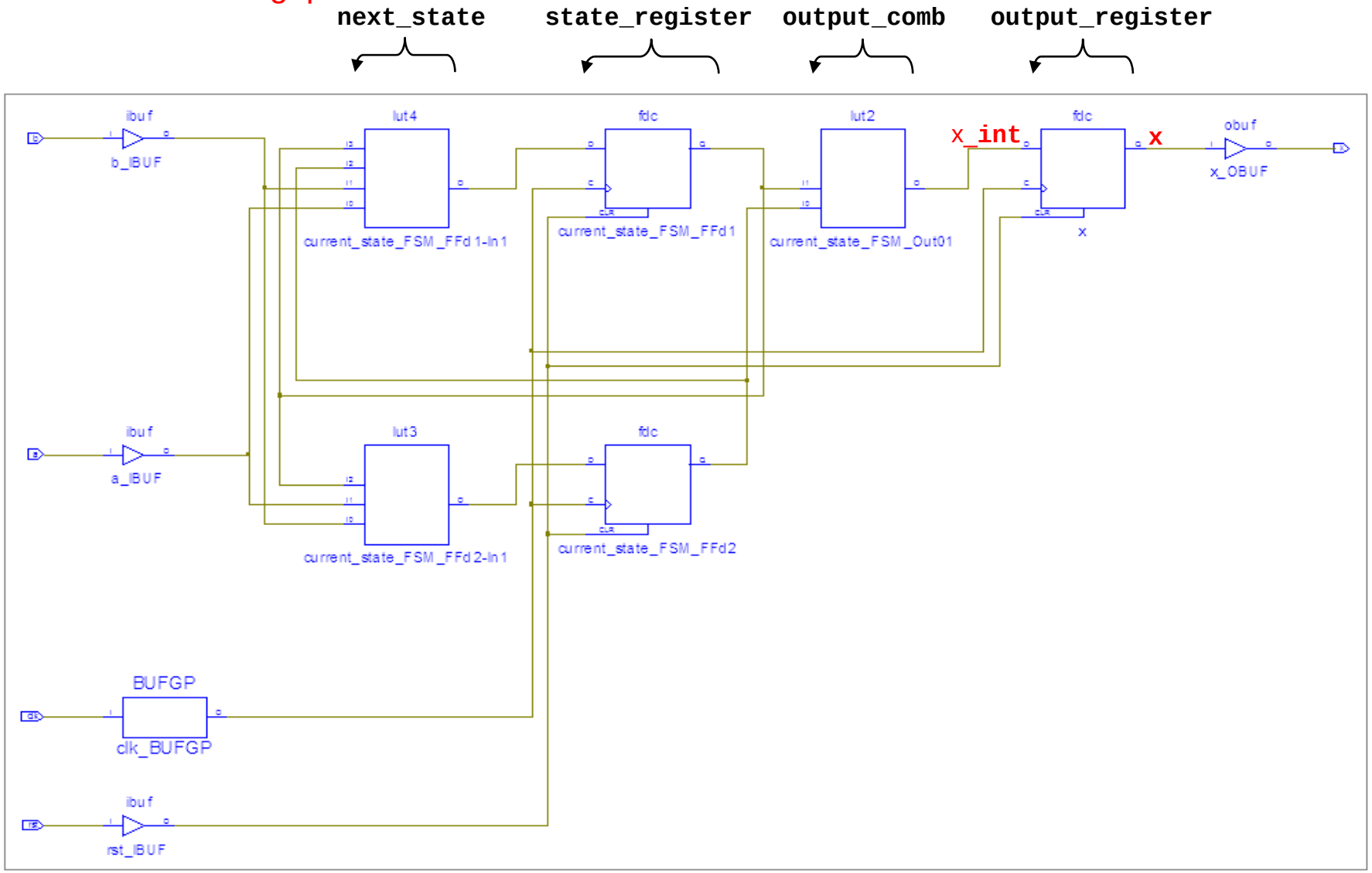


La sortie **x**

- prend la valeur de **x_int** sur un front d'horloge

  ⇨ la sortie **x** est en retard d'une période d'horloge par rapport à l'entrée dans l'état interne **s2**

❑ Schéma technologique



Avantages des assignations des sorties par registre

- Toutes les sorties commutent en même temps (sur le même front d'horloge). A contrario, des sorties à assignations combinatoires risqueraient de changer d'état à des instants différents (en raison de délais de propagation différents dans le bloc combinatoire de sortie).

- En fonction du codage de l'état interne, des transitions multiples pourraient survenir (00 à 11, 11 à 00, 01 à 10 ou 10 à 01) qui généreraient des états transitoires car les flip-flops ne changent pas d'état simultanément ; d'où également la possibilité d'un état transitoire '1' intempestif sur le signal **`x_int`** ; le registre de sortie permet alors de filtrer ce signal parasite.

4.7.3. Action de transition – assignation combinatoire de la sortie

Global Actions **Concurrent Statements** **Architecture Declarations** **Signal Status** **State Register Statements** **Process Declarations**

Pre Actions:

Post Actions:

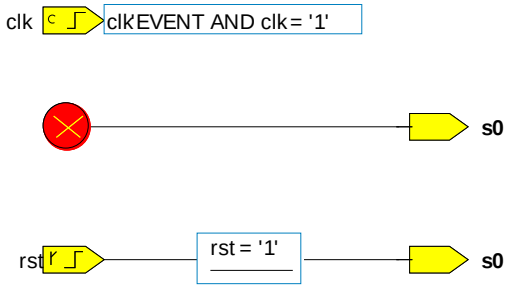
SIGNAL	MODE	DEFAULT	RESET	SCHEME
x	OUT	'0'		COMB

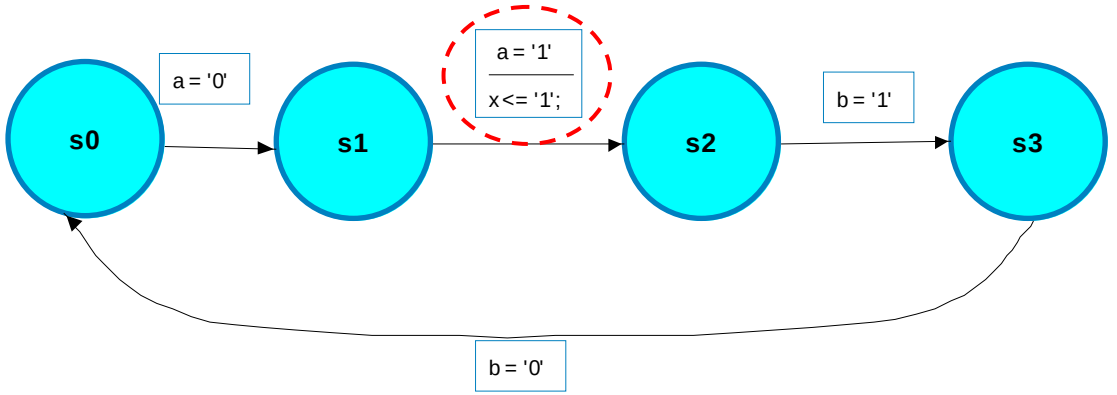
Clocked Process:

Output Process:

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```





<company name>		Project:	mealy_comb
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/fsm/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 05 févr. 2010		

```
output_proc : PROCESS (a, current_state)

   BEGIN
      -- Default Assignment
      x <= '0';
      -- Combined Actions
      CASE current_state IS
         WHEN s1 =>
            IF (a = '1') THEN
               x <= '1';
            END IF;
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS output_proc;
```

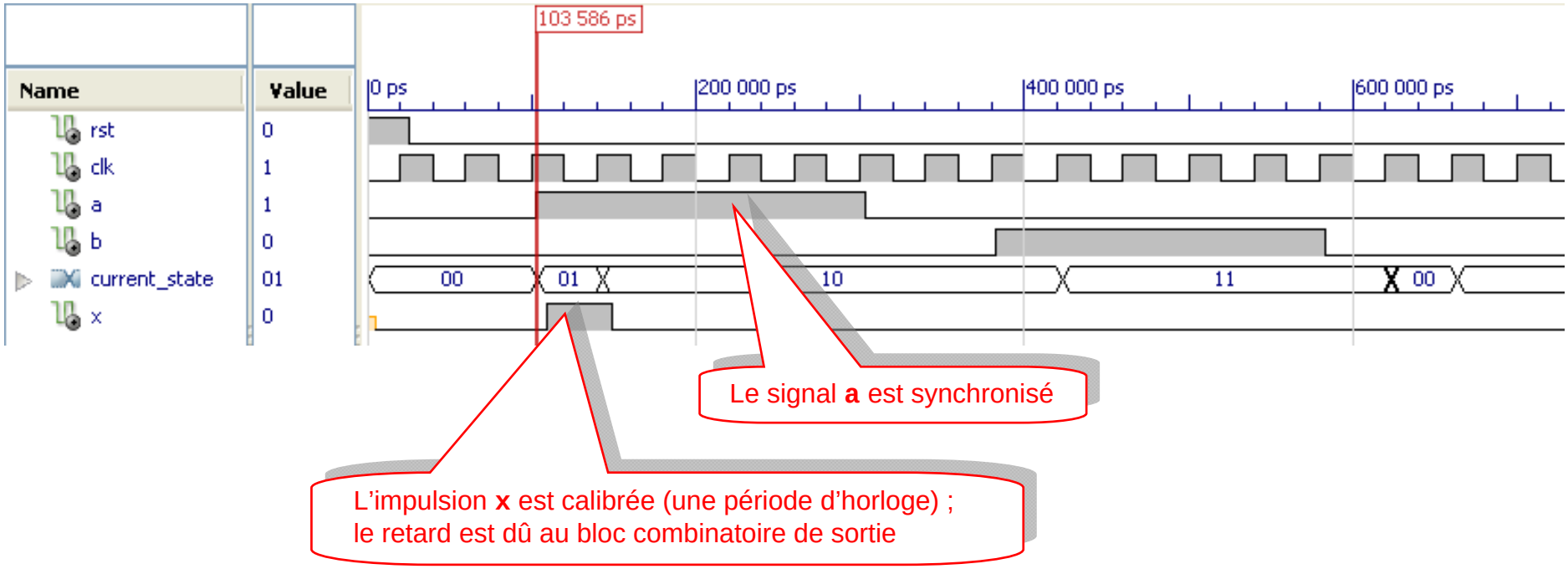
La sortie **x**

- dépend de l'état interne de la machine et de l'entrée **a**
- est à '0' par défaut lorsqu'elle n'est pas assignée explicitement dans un état du graphe
- passe à '1' dès que le signal **a** passe à '1' alors que la machine est dans l'état **s1** (au retard près du bloc combinatoire)
- repasse à '0' dès que la machine entre dans l'état **s2**

Deux cas de figure

- ☹ Le signal d'entrée **a** est asynchrone (mauvaise conception) ➔ l'impulsion **x** a une durée imprévisible !
- ☺ Le signal d'entrée **a** est synchronisé (bonne conception) ➔ l'impulsion **x** a une durée toujours égale à la période d'horloge

Simulation temporelle



Applications courantes des actions de transition

Comme le signal de sortie **x** n'est à '1' que pour un seul front montant d'horloge, il peut servir de signal de validation d'une opération séquentielle ponctuelle :

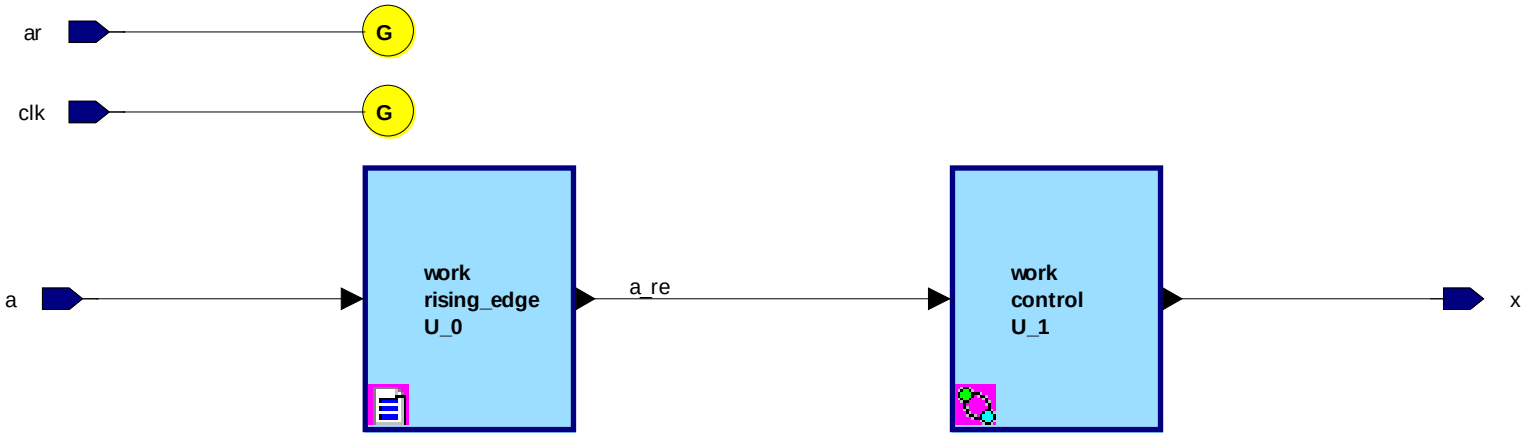
- Incrémenter ou décrémenter un compteur une seule fois, à un moment précis
- Décaler un registre à droite ou à gauche d'une seule position, à un moment précis
- Initialiser ou charger un registre

4.8. Exemples de machines d'état

4.8.1. Transition sur un front montant de signal

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

Declarations
Ports:
```
a    : std_logic
ar   : std_logic
clk  : std_logic
x    : std_logic
```
Diagram Signals:
```
SIGNAL a_re : std_logic
```



<company name>		Project:	rising_edge
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	work/fbd/struct		
Edited:	by UNIVERSITE Hte ALSAC on 08 févr. 2010		

Code VHDL du bloc `rising_edge`

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY rising_edge IS PORT(
      a    : IN     std_logic;
      ar   : IN     std_logic;
      clk  : IN     std_logic;
      a_re : OUT    std_logic);
END rising_edge ;


ARCHITECTURE behavior OF rising_edge IS
     signal a_sy1, a_sy2 : std_logic;
BEGIN

  process (ar,clk)
    begin
      if ar = '1' then
        a_sy1 <= '0';
        a_sy2 <= '0';
      elsif clk'event and clk = '1' then
        a_sy1 <= a;
        a_sy2 <= a_sy1;
      end if;
    end process;

    a_re <= a_sy1 and (not a_sy2);

END ARCHITECTURE behavior;
```

Diagramme d'état du bloc `control`

Global Actions
Pre Actions:
Post Actions:

Concurrent Statements

Architecture Declarations

Signal Status

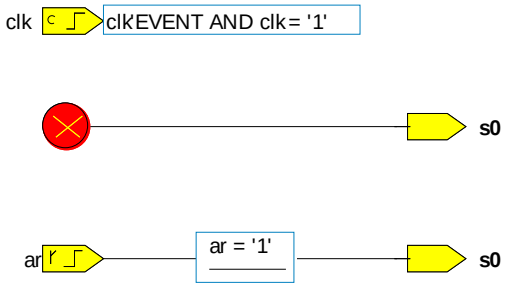
SIGNAL	MODE	DEFAULT	RESET	SCHEME
x	OUT	'0'		COMB

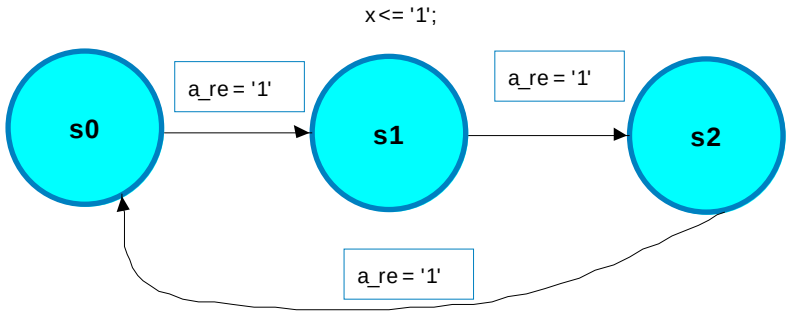
State Register Statements

Process Declarations
Clocked Process:
Output Process:

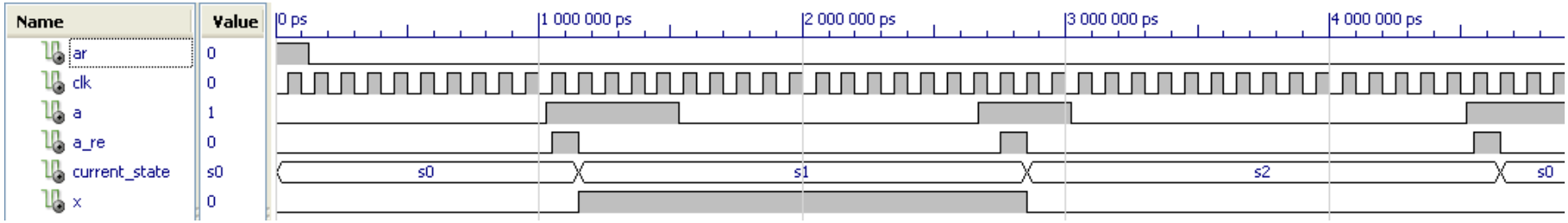
Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```





<company name>		Project:	rising_edge
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/control/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 12 févr. 2010		



Architecture VHDL de la machine d'état `control` (à l'aide de trois processus)

```
ARCHITECTURE fsm OF control IS

   TYPE STATE_TYPE IS (s0, s1, s2);

   -- State vector declaration
   ATTRIBUTE state_vector : string;
   ATTRIBUTE state_vector OF fsm : ARCHITECTURE IS "current_state";

   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;

BEGIN

   ----------------------------------------------------------------
   clocked_proc : PROCESS (clk, ar)
   ----------------------------------------------------------------
   BEGIN
      IF (ar = '1') THEN
         current_state <= s0;
      ELSIF (clk'EVENT AND clk = '1') THEN
         current_state <= next_state;
      END IF;
   END PROCESS clocked_proc;
```

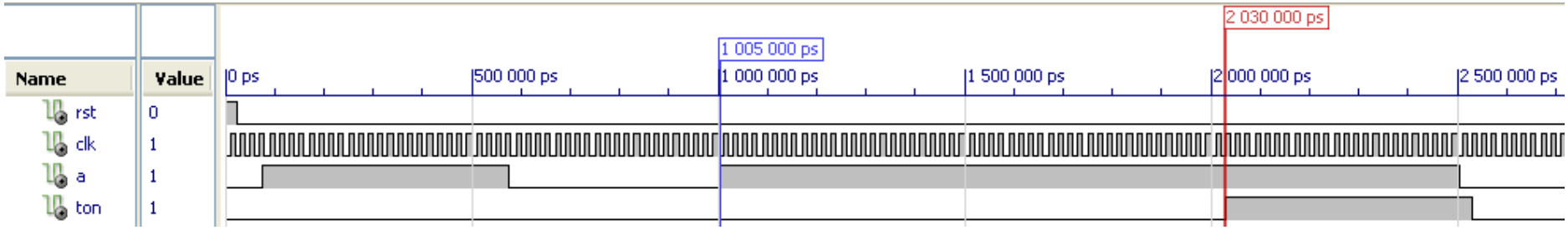
```
-------------------------------------------------------------------
nextstate_proc : PROCESS (a_re, current_state)
-------------------------------------------------------------------
BEGIN
   CASE current_state IS
      WHEN s0 =>
         IF (a_re = '1') THEN
            next_state <= s1;
         ELSE
            next_state <= s0;
         END IF;
      WHEN s1 =>
         IF (a_re = '1') THEN
            next_state <= s2;
         ELSE
            next_state <= s1;
         END IF;
      WHEN s2 =>
         IF (a_re = '1') THEN
            next_state <= s0;
         ELSE
            next_state <= s2;
         END IF;
      WHEN OTHERS =>
         next_state <= s0;
   END CASE;
END PROCESS nextstate_proc;
```

```
-------------------------------------------------------------------
output_proc : PROCESS (current_state)
-------------------------------------------------------------------
BEGIN
   -- Default Assignment
   x <= '0';

   -- Combined Actions
   CASE current_state IS
      WHEN s1 =>
         x <= '1';
      WHEN OTHERS =>
         NULL;
   END CASE;
END PROCESS output_proc;
```

4.8.2. Temporisateur (action de transition et action d'état simples)

Caractéristiques du temporisateur

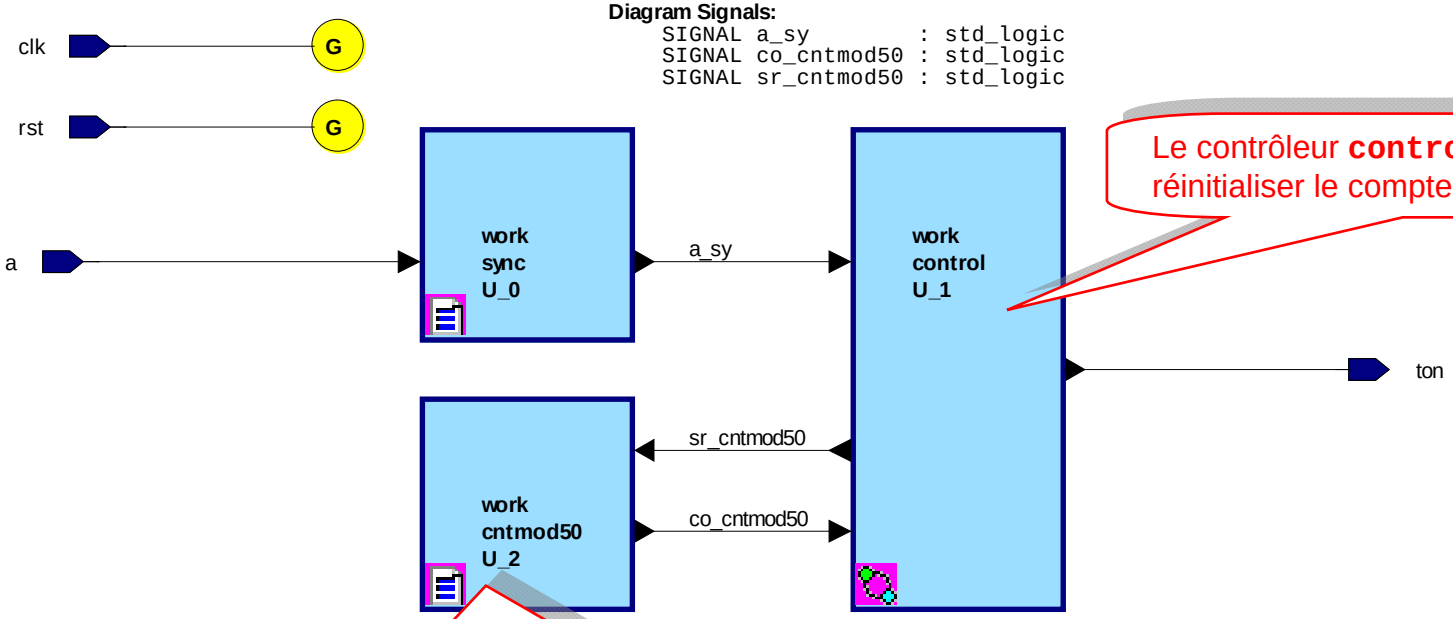
- Fréquence d'horloge : 50 MHz
- Signal de déclenchement de temporisation (asynchrone) : **a**
- Signal de fin de temporisation : **ton**
- Durée de temporisation : 1 µs à 40 ns près



Structure interne du temporisateur

Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

Declarations
Ports:
```
a           : std_logic
clk         : std_logic
rst         : std_logic
ton         : std_logic
```
Diagram Signals:
```
SIGNAL a_sy        : std_logic
SIGNAL co_cntmod50 : std_logic
SIGNAL sr_cntmod50 : std_logic
```



Le contrôleur **`control`** ne fait que réinitialiser le compteur externe **`cntmod50`**

Le compteur modulo 50 **`cntmod50`** compte en permanence.
Le signal **`co_cntmod50`** vaut '1' lorsqu'il contient la valeur 49.

<company name>		Project:	fsm_timer
Title:	<enter diagram title here>	<enter comments here>	
Path:	work/fbd/struct		
Edited:	by UNIVERSITE Hte ALSAC on 13 févr. 2010		

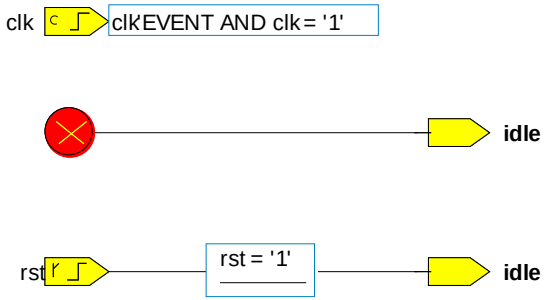
Global Actions **Concurrent Statements** **Architecture Declarations**
Pre Actions:
Post Actions:

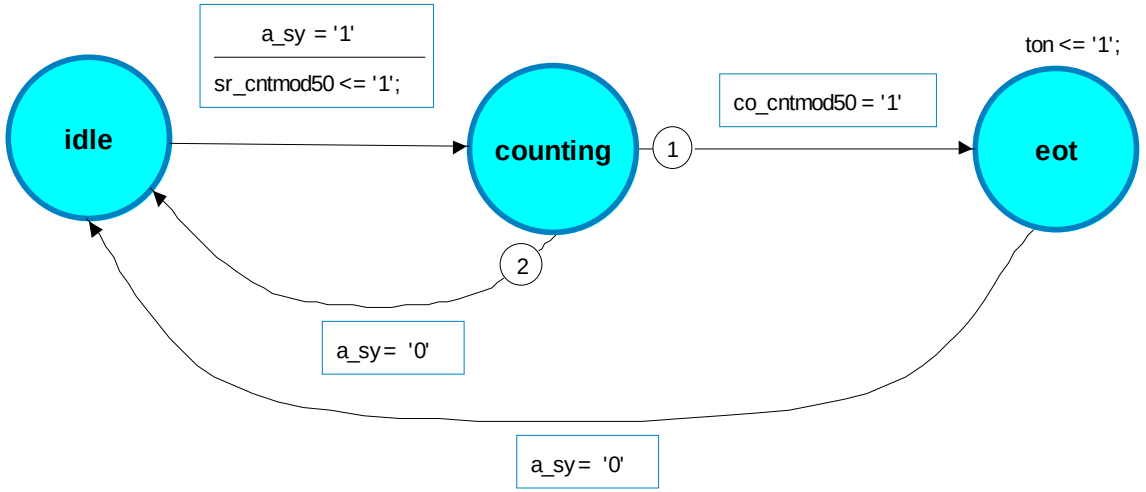
Signal Status **State Register Statements** **Process Declarations**

SIGNAL	MODE	DEFAULT	RESET	SCHEME
sr_cntmod50	OUT	'0'		COMB
ton	OUT	'0'		COMB

Clocked Process:
Output Process:

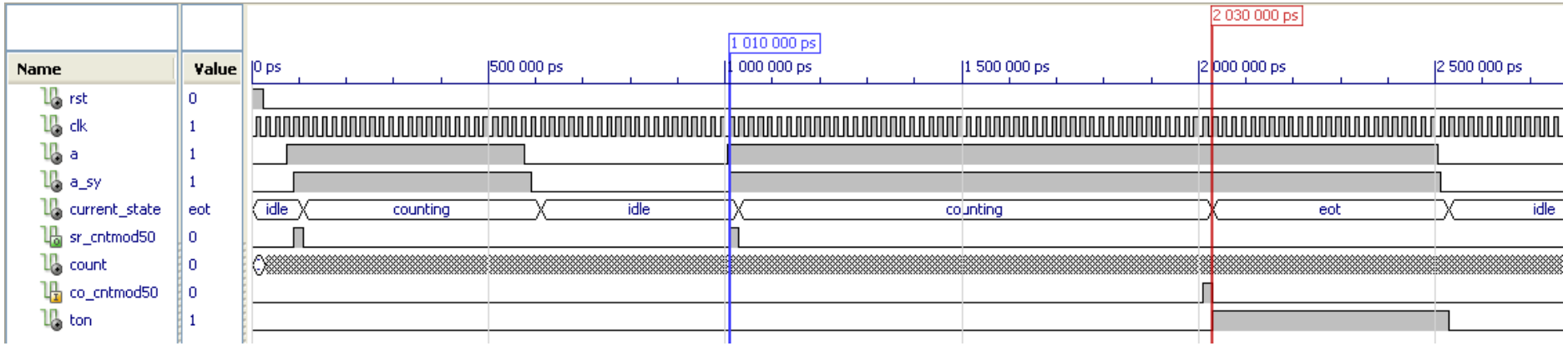
Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

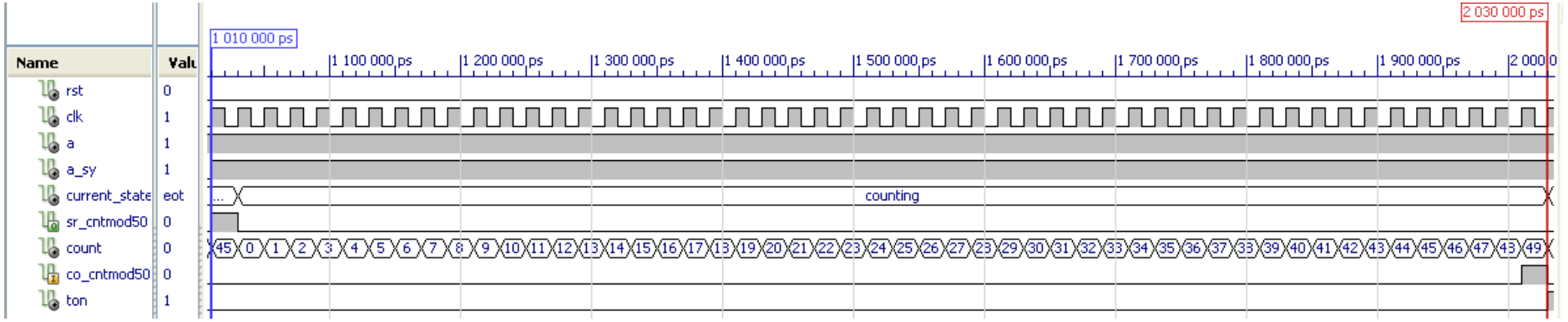




<company name>		Project:	fsm_timer
		<enter comments here>	
Title:	<enter diagram title here>		
Path:	work/control/fsm		
Edited:	by UNIVERSITE Hte ALSAC on 13 févr. 2010		

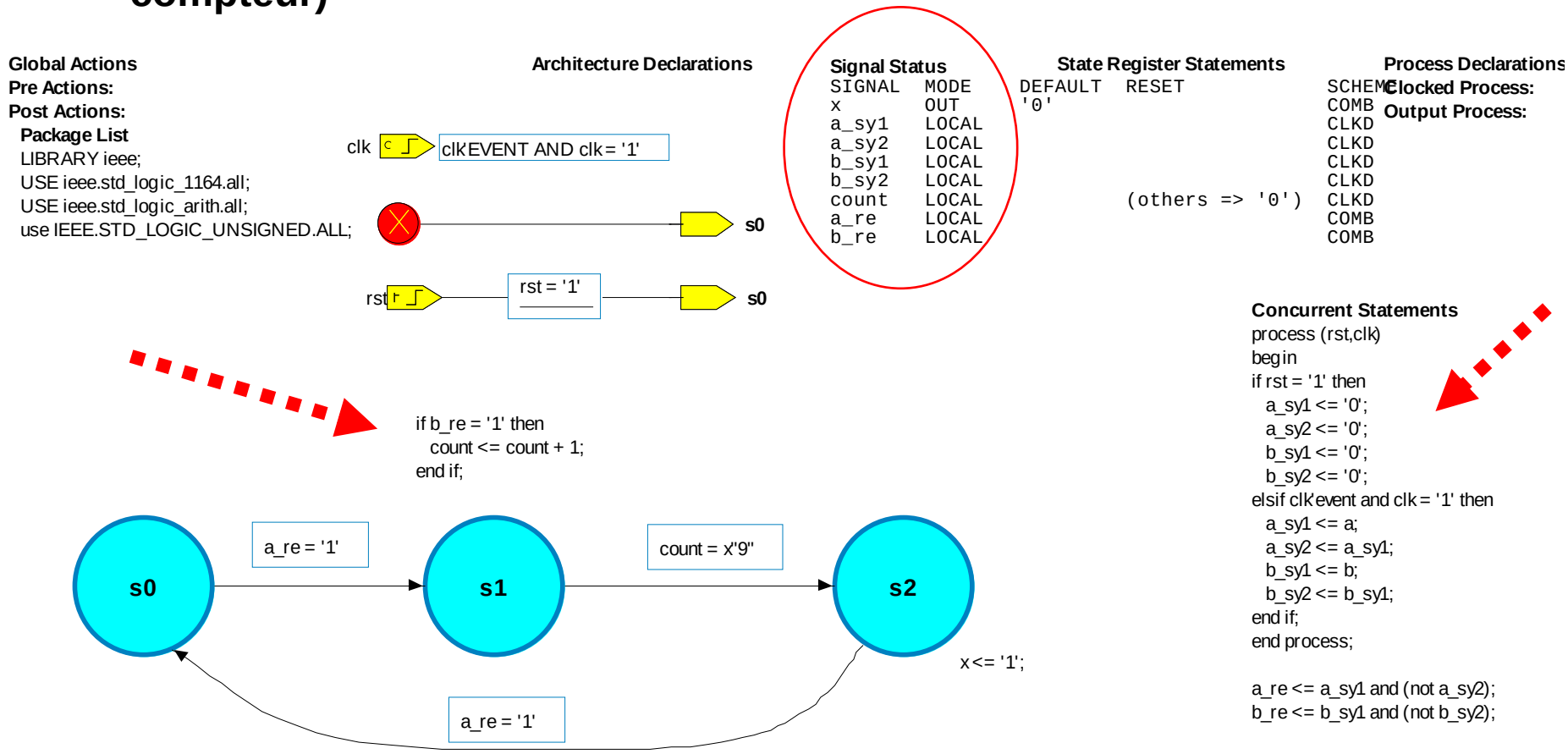
- **`sr_cntmod50`** est le signal de mise à zéro synchrone du compteur **`cntmod50`**. Ce signal est fabriqué par une action de transition
- La fin de temporisation est marquée par le signal **`ton`** fabriqué par une action d'état



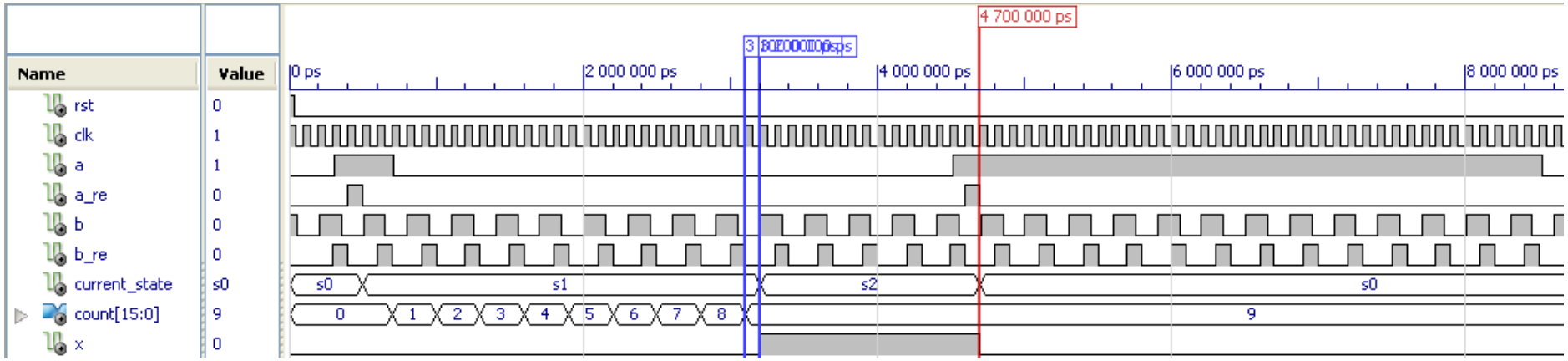


`count` représente la position courante du compteur **`cntmod50`**

4.8.3. Action d'état complexe (incrémentation conditionnelle d'un compteur)



- La description est effectuée entièrement avec l'éditeur de diagramme d'état
- Les détections des fronts montants des signaux d'entrée sont faites dans la zone **`Concurrent Statements`**




```
clocked_proc : PROCESS (clk, rst)
BEGIN
   IF (rst = '1') THEN
      current_state <= s0;
      -- Default Reset Values
      count <= (others => '0');
   ELSIF (clk'EVENT AND clk = '1') THEN
      current_state <= next_state;
      -- Combined Actions
      CASE current_state IS
         WHEN s1 =>
            if b_re = '1' then
               count <= count + 1;
            end if;
         WHEN OTHERS =>
            NULL;
      END CASE;
   END IF;
END PROCESS clocked_proc;
```

La gestion du compteur **count** est décrite dans le même processus (clocked) que le registre d'état de la machine d'état